Progress DataDirect for ODBC for Microsoft SQL Server Wire Protocol Driver User's Guide Release 8.0.2

Generated on: 17 March 2025

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for ODBC for SQL Server Wire Protocol Driver

The Progress DataDirect *for* ODBC SQL Server Wire Protocol driver (the SQL Server Wire Protocol driver) provides read-write access to the following data sources:

Cloud:

- Microsoft Azure Synapse Analytics
- Microsoft Windows Azure SQL Database

On premise:

- Microsoft Analytics Platform System
- Microsoft SQL Server

The documentation for the driver also includes the *Progress DataDirect for ODBC Drivers Reference*. The reference provides general reference information for all DataDirect drivers for ODBC, including content on troubleshooting, supported SQL escapes, and DataDirect tools. For the complete documentation set, visit to the Progress DataDirect Connectors Documentation Hub: https://docs.progress.com/bundle/datadirect-connectors/page/DataDirect-Connectors-by-data-source.html.

Related Links

For details, see the following topics:

- What's new in this release?

What's new in this release?

Support and Certifications

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

- Release Notes: https://www.progress.com/odbc/release-history/
- DataDirect Product Compatibility Guide: https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf

Changes Since 8.0.2 GA

- **Driver Enhancements**
  - The driver is now compiled with a Visual Studio 2022 compiler for the Windows platforms. As a result, you must have Microsoft Visual C/C++ runtime version 14.40.33810 or higher on your machine to run the driver.
  - The driver has been enhanced to support PKCS #12 files as keystore providers for the Always Encrypted feature. PKCS #12 files can be stored locally on the disk on any platform the driver supports. You can configure the driver to use a PKCS #12 file using the new AE Keystore Location and AE Keystore Secret connection options. See Using keystore providers for details.
  - The driver has been enhanced to support Microsoft Entra ID authentication using managed identities for Azure resources. You can configure this feature using the updated Authentication Method (`AuthenticationMethod`) and User Name (`LogonID`) options. See Microsoft Entra ID authentication for details.
  - The driver has been enhanced to support Microsoft Entra ID authentication using service principal users. You can configure this feature using the updated Authentication Method (`AuthenticationMethod`), User Name (`LogonID`) and Password (`Password`) options. See Microsoft Entra ID authentication 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 Encryption Method (EncryptionMethod) option to the new value `8` (Strict). See Encryption Method for details.
  - The default version of the OpenSSL library has been upgraded to 3.0. As part of this upgrade, earlier versions of the OpenSSL library are no longer supported to provide the best protection for your data. The upgrade is available in the following OpenSSL library files: `xx`openssl30.dll (for Windows) and

*xx*openssl30.so [.sl] (for UNIX/Linux).

The OpenSSL 3.0 library uses a set of shared libraries called providers to implement different types of cryptographic algorithms. The driver supports the following OpenSSL 3.0 providers: FIPS and default. See TLS/SSL server authentication and TLS/SSL client authentication for details.

- The driver has been enhanced to support the Windows certificate store for TLS/SSL server authentication. See TLS/SSL server authentication for details.
- The driver has been enhanced to support TLS/SSL server authentication for the applications deployed in a serverless environment. The driver stores the TLS/SSL certificates in memory and lets applications use TLS/SSL server authentication without storing the truststore file on the disk. To use this enhancement, specify the content of the certificate in the refreshed Trust Store (`Truststore`) connection option or the new `SQL_COPT_INMEMORY_TRUSTSTORECERT` pre-connection attribute. See Trust Store and Using SQL_COPT_INMEMORY_TRUSTSTORECERT for details.
- A Password Encryption Tool, called ddencpwd, is now included with the product package. It encrypts passwords for secure handling in connection strings and odbc.ini files. At connection, the driver decrypts these passwords and passes them to the data source as required. See Password Encryption Tool (UNIX/Linux only) for details.
- The driver has been enhanced with the new Keep Connection Active (KeepConnectionActive) and Socket Idle Time (SocketIdleTimeCheckInterval) connection options. Together, these options provide you with a method to keep active idle connections to Azure SQL Database, Azure Synapse Analytics through Azure SQL Gateway, or to databases that enforce timeouts for inactivity. See Keep Connection Active and Socket Idle Time for details.
- The driver supports inserts into IDENTITY columns in data replication scenarios. The Enable Replication User connection option (EnableReplicationUser) or the `SQL_COPT_REPLICATION_USER` connection attribute (numeric value `1080`) can be used to allow inserts into IDENTITY columns defined as NOT FOR REPLICATION. For details, see Inserts on IDENTITY columns in data replication scenarios and Enable Replication User.
- The driver has been enhanced to support distributed transactions. It implements the XA interface to enable support for distributed transactions. For details, see XA interface support.
- The driver is enhanced to support access token authentication programmatically with the use of a pre-connection `SQL_COPT_SS_ACCESS_TOKEN` attribute. See Access token authentication for details.
- The drivers using base version B0649 and later have been enhanced to include timestamp in the internal packet logs by default. If you want to disable the timestamp logging in packet logs, set `PacketLoggingOptions=1`. The internal packet logging is not enabled by default. To enable it, set `EnablePacketLogging=1`.
- The Driver Manager for UNIX/Linux has been enhanced to support setting the Unicode encoding type for applications on a per connection basis. By passing a value for the SQL_ATTR_APP_UNICODE_TYPE attribute using SQLSetConnectAttr, your application can specify the encoding at connection. This allows your application to pass both UTF-8 and UTF-16 encoded strings with a single environment handle.

  The valid values for the SQL_ATTR_APP_UNICODE_TYPE attribute are SQL_DD_CP_UTF8 and SQL_DD_CP_UTF16. The default value is SQL_DD_CP_UTF8.

  Refer to the "Driver Manager and Unicode encoding on UNIX/Linux" in *Progress DataDirect for ODBC Drivers Reference* for details.

  This enhancement is available in build 08.02.0449 of the driver manager.

- The new AllowedOpenSSLVersions option allows you to determine which version of the OpenSSL library file the driver uses for data encryption.
- 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 in encrypted columns and transparently encrypts values bound to SQL parameters or decrypts values returned in results and output parameters.
  - The driver supports configurable caching of column encryption keys for improved performance.
  - The driver supports using Windows Certificate Store and Azure Key Vault as keystore providers.

  You can enable support for Always Encrypted using the new Column Encryption, Key Store Principal, Key Store Secret, and Key Cache Time To Live connection options. See Always Encrypted for details.

- **Changed Behavior**
  - Microsoft has rebranded Azure Active Directory as Microsoft Entra ID. As a result, the driver documentation has been updated to reflect this name change.
  - The TLSv1.1 and TLSv1.0 cryptographic protocols are now disabled by default and have been removed as selectable values for the Crypto Protocol Version (CryptoProtocolVersion) option on the Setup dialog. These protocols are no longer considered secure and, therefore, are no longer recommended for use. However, the driver still supports TLSv1.1 and TLSv1.0 for legacy servers that do not support more secure protocols. See Crypto Protocol Version for more information.
  - The Allowed OpenSSL Versions (AllowedOpenSSLVersions) connection option has been deprecated as the driver currently supports only version 3.0 of the OpenSSL library.
  - The product no longer includes version 1.1.1 of the OpenSSL library. The library will reach the end of its product life cycle in September 2023 and will not receive any security updates after that. Note that continuing to use the library after September 2023 can potentially expose you to security vulnerabilities.

    Note: As a result of this change, when installing a new version of the product, the installer program will automatically remove version 1.1.1 of the library from the install directory, which will impact all the DataDirect ODBC drivers installed on a machine. Therefore, if you are using multiple drivers, upgrade all your drivers to the latest version.

  - The product no longer includes version 1.0.2 of the OpenSSL library. The library has reached the end of its product life cycle and is not receiving security updates anymore. Note that continuing to use the library could potentially expose you to security vulnerabilities.

    Note: As a result of this change, when installing a new version of the driver, the installer program will automatically remove version 1.0.2 of the library from the install directory.

  - The crypto protocol versions prior to TLSv1 are no longer supported.
  - When executing SQLGetInfo(SQL_COLLATION_SEQ), the driver now fetches the collation sequence from the server. Earlier, it always returned ISO 8859-1 as the collation sequence.
  - The driver has been updated to return the server name to which you are connected for the value of `SQL_SERVER_NAME` when executing SQLGetInfo. In earlier versions of the driver, the value returned for `SQL_SERVER_NAME` would be the setting of the Host Name (`HostName`) connection option. The driver will now return the server name string that it receives from the server when connecting to the database as the value for `SQLGetInfo(SQL_SERVER_NAME)`.
  - The following Windows platforms have reached the end of their product lifecycle and are no longer

supported by the driver:
- Windows 8.0 (versions 8.1 and higher are still supported)
- Windows Vista (all versions)
- Windows XP (all versions)
- Windows Server 2003 (all versions)

Changes for 8.0.2 GA

- **Platform Certifications**
  - Certified with Red Hat Enterprise 7.3
  - Certified with Debian 7.11, 8.5
  - Certified with Ubuntu 14.04, 16.04
  - Certified with Windows Server 2016
- **Data Source Version Certifications**
  - Certified with Microsoft Azure Synapse Analytics 12.00
  - Certified with Microsoft Analytics Platform System 10.00
  - Certified with Microsoft SQL Server 2016, 2017
- **Driver Enhancements**
  - The driver has been enhanced to transparently connect to Microsoft Azure Synapse Analytics and Microsoft Analytics Platform System data sources. See Support for Azure Synapse Analytics and Analytics Platform System for more information about supported features and functionality.
  - The driver has been enhanced to support connecting to a proxy server through an HTTP connection. HTTP proxy support is configurable with five new connection options. See Proxy Host, Proxy Mode, Proxy Password, Proxy Port, and Proxy User for details.
  - The new Enable Server Side Cursors connection option allows you to determine which server-side cursors are enabled for the data source. See Enable Server Side Cursors for details.
  - 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 for Azure SQL Database that allows you to centrally manage identities of database users. See Microsoft Entra ID authentication for details.
  - 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. To support this enhancement, the following updates have been made to the driver:
    - The new Multi-Subnet Failover option allows the driver to attempt parallel connections to all the IP addresses associated with an availability group when the primary listener is unavailable. This offers improved response time over traditional failover, which attempts connections to alternate servers one at a time. To support high availability with Always On, this option must be enabled.
    - The Host Name option has been updated to support the virtual network name (VNN) of the availability group listener as a valid value. To connect to an Always On Availability group, you must specify the VNN using this option.
    - The new Application Intent option allows you to control whether the driver requests read-only routing, thereby improving efficiency by reducing the workload on read-write nodes.

    See Multi-Subnet Failover, Host Name, and Application Intent for details.
  - The driver and Driver Manager have been enhanced to support UTF-8 encoding in the odbc.ini and

odbcinst.ini files.

Refer to the "Character encoding in the odbc.ini and odbcinst.ini files" in *Progress DataDirect for ODBC Drivers Reference* for details.

- **Changed Behavior**
  - The default value for Crypto Protocol Version has been updated to `TLSv1.3,TLSv1.2`. This change improves the security of the driver by employing only the most secure cryptographic protocols as the default behavior. See Crypto Protocol Version for details.
  - The default value for Crypto Protocol Version has been updated to `TLSv1.2,TLSv1.1,TLSv1`. This change improves the security of the driver by employing only the most secure cryptographic protocols as the default behavior. See Crypto Protocol Version for details.

Driver requirements

Data source and platform requirements

- The SQL Server Wire Protocol driver connects via TCP/IP. TCP/IP connections must be configured on the Windows server on which the Microsoft SQL Server database resides.

For the latest support information, visit the DataDirect Product Compatibility Guide: https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf.

Windows requirements for 32-bit drivers

- All required network software that is supplied by your database system vendors must be 32-bit compliant.
- You must have Microsoft Visual C/C++ runtime version 14.40.33810 or higher.
- You must have ODBC header files to compile your application. For example, Microsoft Visual Studio includes these files.

Windows requirements for 64-bit drivers

- All required network software that is supplied by your database system vendors must be 64-bit compliant.
- You must have Microsoft Visual C/C++ runtime version 14.40.33810 or higher.
- You must have ODBC header files to compile your application. For example, Microsoft Visual Studio includes these files.

Linux requirements for 32-bit drivers

- If your application was built with 32-bit system libraries, you must use 32-bit drivers. The database to which you are connecting can be either 32-bit or 64-bit enabled.
- An application compatible with components that were built using g++ GNU project C++ Compiler version 3.4.6 and the Linux native pthread threading model (Linuxthreads).

Linux requirements for 64-bit drivers

- An application compatible with components that were built using g++ GNU project C++ Compiler version 3.4 and the Linux native pthread threading model (Linuxthreads).

AIX requirements for 32-bit and 64-bit drivers

- IBM POWER processor
- An application compatible with components that were built using Visual Age C++ 6.0.0.0 and the AIX native threading model.

HP-UX requirements for 32-bit drivers

- The following processors are supported:
  - PA-RISC
  - Intel Itanium II (IPF)
- For PA-RISC: An application compatible with components that were built using HP aC++ 3.30 and the HP-UX 11 native (kernel) threading model (posix draft 10 threads).
- For IPF: An application compatible with components that were built using HP aC++ 5.36 and the HP-UX 11 native (kernel) threading model (posix draft 10 threads).

HP-UX requirements for 64-bit drivers

- Intel Itanium II (IPF) processor
- HP aC++ v. 5.36 and the HP-UX 11 native (kernel) threading model (posix draft 10 threads).

Oracle Solaris requirements for 32-bit drivers

- The following processors are supported:
  - Oracle SPARC
  - x86: Intel
  - x64: Intel and AMD
- For Oracle SPARC: An application compatible with components that were built using Oracle Workshop version 6 update 2 and the Solaris native (kernel) threading model.
- For x86/x64: An application compatible with components that were built using Oracle C++ 5.8 and the Solaris native (kernel) threading model.

Oracle Solaris requirements for 64-bit drivers

- The following processors are supported:
  - Oracle SPARC

  - x64: Intel and AMD

- For Oracle SPARC: An application compatible with components that were built using Oracle Workshop version 6 update 2 and the Solaris native (kernel) threading model.
- For x64: An application compatible with components that were built using Oracle C++ Compiler version 5.8 and the Solaris native (kernel) threading model.

ODBC compliance

The SQL Server Wire Protocol driver is compliant with the Open Database Connectivity (ODBC) specification. The driver supports ODBC conformance level 1.

In addition, the following functions are supported:

- SQLForeignKeys
- SQLTablePrivileges
- SQLDescribeParam
- SQLColumnPrivileges

Refer to "ODBC API and scalar functions" in the *Progress DataDirect for ODBC Drivers Reference* for additional information.

Version string information

The driver has a version string of the format:

XX.YY.ZZZZ(B*AAAA*, U*BBBB*)

or

XX.YY.ZZZZ(b*AAAA*, u*BBBB*)

The Driver Manager on UNIX and Linux has a version string of the format:

XX.YY.ZZZZ(U*BBBB*)

The component for the Unicode conversion tables (ICU) has a version string of the format:

XX.YY.ZZZZ

where:

XX is the major version of the product.

YY is the minor version of the product.

ZZZZ is the build number of the driver or ICU component.

AAAA is the build number of the driver's bas component.

BBBB is the build number of the driver's utl component.

For example:

```
08.02.0002 (b0001, u0002)
      |__|  |___|  |___|
     Driver  Bas    Utl
```


On Windows, you can check the version string through the properties of the driver DLL. Right-click the driver DLL and select **Properties**. The Properties dialog box appears. On the Version tab, click **File Version** in the Other version information list box.

You can always check the version string of a driver on Windows by looking at the About tab of the driver's Setup dialog.

UNIX® On UNIX and Linux, you can check the version string by using the test loading tool shipped with the product. This tool, ivtestlib for 32-bit drives and ddtestlib for 64-bit drivers, is located in *install_directory*/bin.

The syntax for the tool is:

```
ivtestlib shared_object
```

or

```
ddtestlib shared_object
```

For example, for the 32-bit driver on Linux:

```
ivtestlib ivsqls28.so
```

returns:

```
08.02.0001 (B0002, U0001)
```

For example, for the Driver Manager on Linux:

```
ivtestlib libodbc.so
```

returns:

```
08.02.0001 (U0001)
```

For example, for the 64-bit Driver Manager on Linux:

```
ddtestlib libodbc.so
```

returns:

```
08.02.0001 (U0001)
```

For example, for 32-bit ICU component on Linux:

```
ivtestlib libivicu28.so
08.02.0001
```

Note: On AIX, Linux, and Solaris, the full path to the driver does not have to be specified for the test loading tool. The HP-UX version of the tool, however, requires the full path.

getFileVersionString function

Version string information can also be obtained programmatically through the function `getFileVersionString`. This function can be used when the application is not directly calling ODBC functions.

This function is defined as follows and is located in the driver's shared object:

```
const unsigned char* getFileVersionString();
```

This function is prototyped in the qesqlext.h file shipped with the product.

Data types

The following table shows how the Microsoft SQL Server and Windows Azure SQL Database data types are mapped to the standard ODBC data types. "Unicode support" lists Microsoft SQL Server to Unicode data type mappings.

Table 1: Microsoft SQL Server Data Types

SQL Server	ODBC
bigint	SQL_BIGINT
bigint identity	SQL_BIGINT
binary	SQL_BINARY
bit	SQL_BIT
char	SQL_CHAR
date[1]	SQL_TYPE_DATE
datetime	SQL_TYPE_TIMESTAMP
datetime2[1]	SQL_TYPE_TIMESTAMP
datetimeoffset[2, 3]	SQL_VARCHAR

SQL Server	ODBC
decimal	SQL_DECIMAL
decimal() identity	SQL_DECIMAL
float	SQL_FLOAT
image	SQL_LONGVARBINARY
int	SQL_INTEGER
int identity	SQL_INTEGER
money	SQL_DECIMAL
numeric	SQL_NUMERIC
numeric() identity	SQL_NUMERIC
real	SQL_REAL
smalldatetime	SQL_TYPE_TIMESTAMP
smallint	SQL_SMALLINT
smallint identity	SQL_SMALLINT
smallmoney	SQL_DECIMAL
text	SQL_LONGVARCHAR
time[1, 4]	SQL_TYPE_TIMESTAMP
timestamp	SQL_BINARY
tinyint	SQL_TINYINT
tinyint identity	SQL_TINYINT
uniqueidentifier	SQL_GUID
varbinary	SQL_VARBINARY
varbinary(max)[5]	SQL_LONGVARBINARY
varchar	SQL_VARCHAR
varchar(max)[5]	SQL_LONGVARCHAR

[1] Supported only on Microsoft SQL Server 2008 and higher.

[2] Supported only for Microsoft SQL Server 2008 and higher, and Windows Azure SQL Database.

[3] Datetimeoffset mapping changes based on the setting of the Fetch TSWTZ as Timestamp option.

[4] Time mapping changes based on the setting of the Fetch TWFS as Time option.

[5] Supported only on Microsoft SQL Server 2005 and higher.

Unicode support

The SQL Server Wire Protocol driver maps the Microsoft SQL Server and Windows Azure SQL Database data types to Unicode data types as shown in the following table:

Table 2: Mapping Microsoft SQL Server and Windows Azure SQL Database Data Types to Unicode Data Types

SQL Server Data Type	Mapped to. . .
nchar	SQL_WCHAR
ntext	SQL_WLONGVARCHAR
nvarchar	SQL_WVARCHAR
nvarchar(max)[1]	SQL_WLONGVARCHAR
sysname	SQL_WVARCHAR
xml[1]	SQL_WLONGVARCHAR

The driver supports the Unicode ODBC W (Wide) function calls, such as SQLConnectW. This allows the Driver Manager to transmit these calls directly to the driver. Otherwise, the Driver Manager would incur the additional overhead of converting the W calls to ANSI function calls, and vice versa.

See "UTF-16 applications on UNIX and Linux" for related details.

Also, refer to "Internationalization, localization, and Unicode" in the *Progress DataDirect for ODBC Drivers Reference* for a more detailed explanation of Unicode.

[1] nvarchar(max) and xml are supported for Microsoft SQL Server 2005 and higher.

Using the XML data type

By default, Microsoft SQL Server returns XML data to the driver encoded as UTF-8. To avoid data loss, an application must bind XML data as SQL_C_WCHAR. The driver then returns the data as either UTF-8 or UTF-16, depending on platform and application settings. If the application binds XML data as SQL_C_CHAR, the driver converts it to the client character encoding, possibly causing data loss or corruption. To prevent any conversion of XML data, the application must set the attribute "XML describe type" to SQL_LONGVARBINARY (-10) and bind the data as SQL_C_BINARY.

Retrieving data type information

At times, you might need to get information about the data types that are supported by the data source, for example, precision and scale. You can use the ODBC function SQLGetTypeInfo to do this.

On Windows, you can use ODBC Test to call SQLGetTypeInfo against the ODBC data source to return the data type information.

Refer to "Diagnostic tools" in the *Progress DataDirect for ODBC Drivers Reference* for details about ODBC

Test.

On all platforms, an application can call SQLGetTypeInfo. Here is an example of a C function that calls SQLGetTypeInfo and retrieves the information in the form of a SQL result set.

```
void ODBC_GetTypeInfo(SQLHANDLE hstmt, SQLSMALLINT dataType)
{
    RETCODE rc;

// There are 19 columns returned by SQLGetTypeInfo.
// This example displays the first 3.
// Check the ODBC 3.x specification for more information.
// Variables to hold the data from each column
    char             typeName[30];
    short            sqlDataType;
    unsigned int     columnSize;

    SQLLEN           strlenTypeName,
                     strlenSqlDataType,
                     strlenColumnSize;

    rc = SQLGetTypeInfo(hstmt, dataType);
    if (rc == SQL_SUCCESS) {

// Bind the columns returned by the SQLGetTypeInfo result set.
        rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, &typeName,
              (SDWORD)sizeof(typeName), &strlenTypeName);
        rc = SQLBindCol(hstmt, 2, SQL_C_SHORT, &sqlDataType,
              (SDWORD)sizeof(sqlDataType), &strlenSqlDataType);
        rc = SQLBindCol(hstmt, 3, SQL_C_LONG, &columnSize,
              (SDWORD)sizeof(columnSize), &strlenColumnSize);

// Print column headings
         printf ("TypeName      DataType          ColumnSize\n");
         printf ("--------------------- ---------- ----------\n");

        do {

// Fetch the results from executing SQLGetTypeInfo
            rc = SQLFetch(hstmt);
            if (rc == SQL_ERROR) {
// Procedure to retrieve errors from the SQLGetTypeInfo function
               ODBC_GetDiagRec(SQL_HANDLE_STMT, hstmt);
               break;
              }

// Print the results
         if ((rc == SQL_SUCCESS) || (rc == SQL_SUCCESS_WITH_INFO)) {
printf ("%-30s %10i %10u\n", typeName, sqlDataType, columnSize);
             }

        } while (rc != SQL_NO_DATA);
    }
}
```

Troubleshooting

The *Progress DataDirect for ODBC Drivers Reference* provides information on troubleshooting problems should they occur. Refer to Troubleshooting 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 ODBC Drivers Reference* or use the links below to view some common topics:

- "Code page values" lists supported code page values, along with a description, for the Progress DataDirect for ODBC drivers.
- "ODBC API and scalar functions" lists the ODBC API functions supported by Progress DataDirect for ODBC drivers. In addition, it documents the scalar functions that you use in SQL statements.
- "Internationalization, localization, and Unicode" provides an overview of how internationalization, localization, and Unicode relate to each other. It also includes a background on Unicode, and how it is accommodated by Unicode and non-Unicode ODBC drivers.
- "Security best practices for ODBC 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.

January 2021, Release 8.0.2 for the Progress DataDirect for ODBC for SQL Server Wire Protocol Driver, Version 0001

2

Getting started

This section provides basic information about configuring your driver immediately after installation, testing your connection, and accessing your data with some commonly used third-party applications. To take full advantage of the features of the driver, read "Using the driver".

Information that the driver needs to connect to a database is stored in a *data source*. The ODBC specification describes three types of data sources: user data sources, system data sources (not a valid type on UNIX/Linux), and file data sources. On Windows, user and system data sources are stored in the registry of the local computer. The difference is that only a specific user can access user data sources, whereas any user of the machine can access system data sources. On Windows, UNIX, and Linux, file data sources, which are simply text files, can be stored locally or on a network computer, and are accessible to other machines.

When you define and configure a data source, you store default connection values for the driver that are used each time you connect to a particular database. You can change these defaults by modifying the data source.

Related Links

For details, see the following topics:

- Configuring and connecting on Windows
- Configuring and connecting on UNIX and Linux

Configuring and connecting on Windows

The following basic information enables you to configure a data source and test connect with a driver immediately after installation. On Windows, you can configure and modify data sources through the ODBC Administrator using a driver Setup dialog box. Default connection values are specified through the options on the tabs of the Setup dialog box and are stored either as a user or system data source in the Windows Registry, or as a file data source in a specified location.

Configuring a data source

To configure a data source:

1. From the Progress DataDirect program group, start the ODBC Administrator and click either the **User DSN**, **System DSN**, or **File DSN** tab to display a list of data sources.
   - **User DSN**: If you installed a default DataDirect ODBC user data source as part of the installation, select the appropriate data source name and click **Configure** to display the driver Setup dialog box.

   If you are configuring a new user data source, click **Add** to display a list of installed drivers. Select the appropriate driver and click **Finish** to display the driver Setup dialog box.

   - **System DSN**: To configure a new system data source, click **Add** to display a list of installed drivers. Select the appropriate driver and click **Finish** to display the driver Setup dialog box.
   - **File DSN**: To configure a new file data source, click **Add** to display a list of installed drivers. Select the driver and click **Advanced** to specify attributes; otherwise, click **Next** to proceed. Specify a name for the data source and click **Next**. Verify the data source information; then, click **Finish** to display the driver Setup dialog box.

   The General tab of the Setup dialog box appears by default.

   Note: The General tab displays only fields that are required for creating a data source. The fields on all other tabs are optional, unless noted otherwise in this book.

   ---

2. On the General tab, provide the following information; then, click **Apply**.
   Host Name: Type either the name or the IP address of the server to which you want to connect:
   - If your network supports named servers, you can specify an address as: *`server_name`*. For example, you can enter `SSserver`.
   - You can also specify a named instance of Microsoft SQL Server. Specify this address as: *`server_name\instance_name`*. If only a server name is specified with no instance name, the driver uses the default named instance on the server.

   Port Number: Type the port number of the server listener. The default port number is `1433`.

   Database: Type the name of the database to which you want to connect by default.

Testing the connection

To test the connection:

1. After you have configured the data source, you can click **Test Connect** on the Setup dialog box to attempt to connect to the data source using the connection options specified in the dialog box. The driver returns a message indicating success or failure. A logon dialog box appears as described in "Using a logon dialog box."
2. Supply the requested information in the logon dialog box and click **OK**. Note that the information you enter in the logon dialog box during a test connect is not saved.
   - If the driver can connect, it releases the connection and displays a `Connection Established` message. Click **OK**.
   - If the driver cannot connect because of an incorrect environment or connection value, it displays an appropriate error message. Click **OK**.
3. On the driver Setup dialog box, click **OK**. The values you have specified are saved and are the defaults used when you connect to the data source. You can change these defaults by using the previously described procedure to modify your data source. You can override these defaults by connecting to the data source using a connection string with alternate values. See "Using a connection string" for information about using connection strings.

Configuring and connecting on UNIX and Linux

UNIX®

The following basic information enables you to configure a data source and test connect with a driver immediately after installation. See "Configuring and connecting to data sources" for detailed information about configuring the UNIX/Linux environment and data sources.

Note: In the following examples, *xx* in a driver filename represents the driver level number.

Environment configuration

To configure the environment:

1. Check your permissions: You must log in as a user with full r/w/x permissions recursively on the entire product installation directory.
2. From your login shell, determine which shell you are running by executing:

   ```
   echo $SHELL
   ```

3. Run one of the following product setup scripts from the installation directory to set variables: odbc.sh or odbc.csh. For Korn, Bourne, and equivalent shells, execute odbc.sh. For a C shell, execute odbc.csh. After running the setup script, execute:

   ```
   env
   ```

   to verify that the *installation_directory*/lib directory has been added to your shared library path.

4. Set the ODBCINI environment variable. The variable must point to the path from the root directory to the system information file where your data source resides. The system information file can have any name, but the product is installed with a default file called odbc.ini in the product installation directory. For example, if you use an installation directory of /opt/odbc and the default system information file, from the Korn or Bourne shell, you would enter:

   ```
   ODBCINI=/opt/odbc/odbc.ini; export ODBCINI
   ```

   From the C shell, you would enter:

   ```
   setenv ODBCINI /opt/odbc/odbc.ini
   ```

Test loading the driver

The ivtestlib (32-bit drivers) and ddtestlib (64-bit drivers) test loading tools are provided to test load drivers and help diagnose configuration problems in the UNIX and Linux environments, such as environment variables not correctly set or missing database client components. This tool is installed in the /bin subdirectory in the product installation directory. It attempts to load a specified ODBC driver and prints out all available error information if the load fails.

For example, if the drivers are installed in /opt/odbc/lib, the following command attempts to load the 32-bit driver, where *xx* represents the version number of the driver:

```
ivtestlib /opt/odbc/lib/ivsqlsxx.so
```

Note: On Solaris, AIX, and Linux, the full path to the driver does not have to be specified for the tool. The HP-UX version, however, requires the full path.

If the load is successful, the tool returns a success message along with the version string of the driver. If the driver cannot be loaded, the tool returns an error message explaining why.

Configuring a data source in the system information file

The default odbc.ini file installed in the installation directory is a template in which you create data source definitions. You enter your site-specific database connection information using a text editor. Each data source definition must include the keyword `Driver=`, which is the full path to the driver.

The following examples show the minimum connection string options that must be set to complete a test connection, where *xx* represents iv for 32-bit or dd for 64-bit drivers, *yy* represents the driver level number, and *zz* represents the extension. The values for the options are samples and are not necessarily the ones you would use.

```
[ODBC Data Sources]
SQL Server Wire Protocol=DataDirect 8.0 SQL Server Wire Protocol

[SQL Server Wire Protocol]
Driver=ODBCHOME/lib/xxsqlsyy.zz
Database=default
HostName=SSServer
PortNumber=1433
```

Connection Option Descriptions:

`Database`: The name of the database to which you want to connect by default.

`HostName`: Either the name or the IP address of the server to which you want to connect:

- If your network supports named servers, you can specify an address as: *`server_name`*. For example, you can enter `SSserver`.
- If your network supports named instances of Microsoft SQL Server, you can specify this address as: *`server_name`*`\`*`instance_name`*. If only a server name is specified with no instance name, the driver uses the default named instance on the server.

`PortNumber`: The port number of the server listener. The default is `1433`.

Testing the connection

The driver installation includes an ODBC application called example that can be used to connect to a data source and execute SQL. The application is located in the *`installation_directory`*/samples/example directory.

To run the program after setting up a data source in the odbc.ini, enter `example` and follow the prompts to enter your data source name, user name, and password. If successful, a `SQL>` prompt appears and you can type in SQL statements such as `SELECT * FROM` *`table`*. If example is unable to connect, the appropriate error message is returned.

3

Tutorials

The following sections guide you through using the driver to access your data with some common third-party applications.

Related Links

For details, see the following topics:

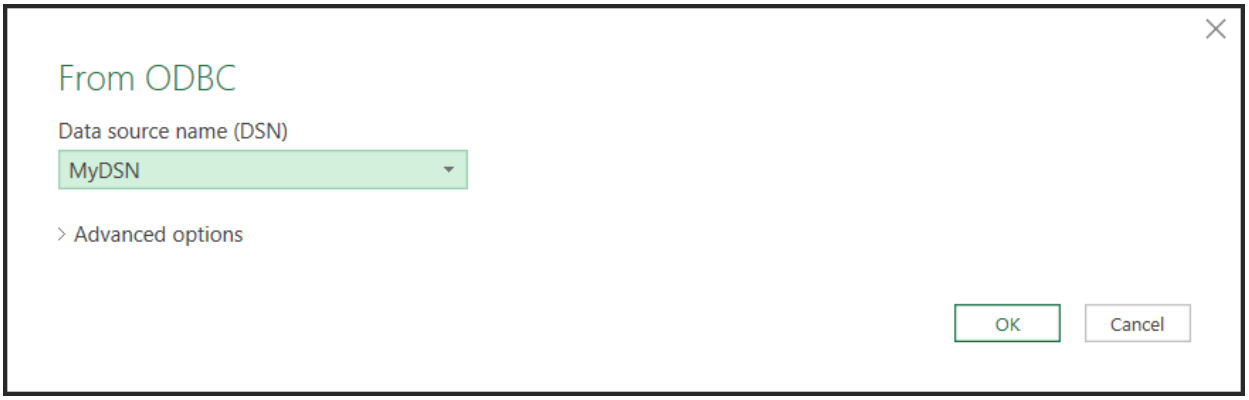
- Accessing data in Microsoft Excel (Windows only)
- Accessing data in Microsoft Excel from the Query Wizard (Windows only)

Accessing data in Microsoft Excel (Windows only)

After you have configured your data source, you can use the driver to access your data with Microsoft Excel from the Data Connection Wizard. Using the driver with Excel provides improved performance when retrieving data, while leveraging the driver's relational-mapping tools.

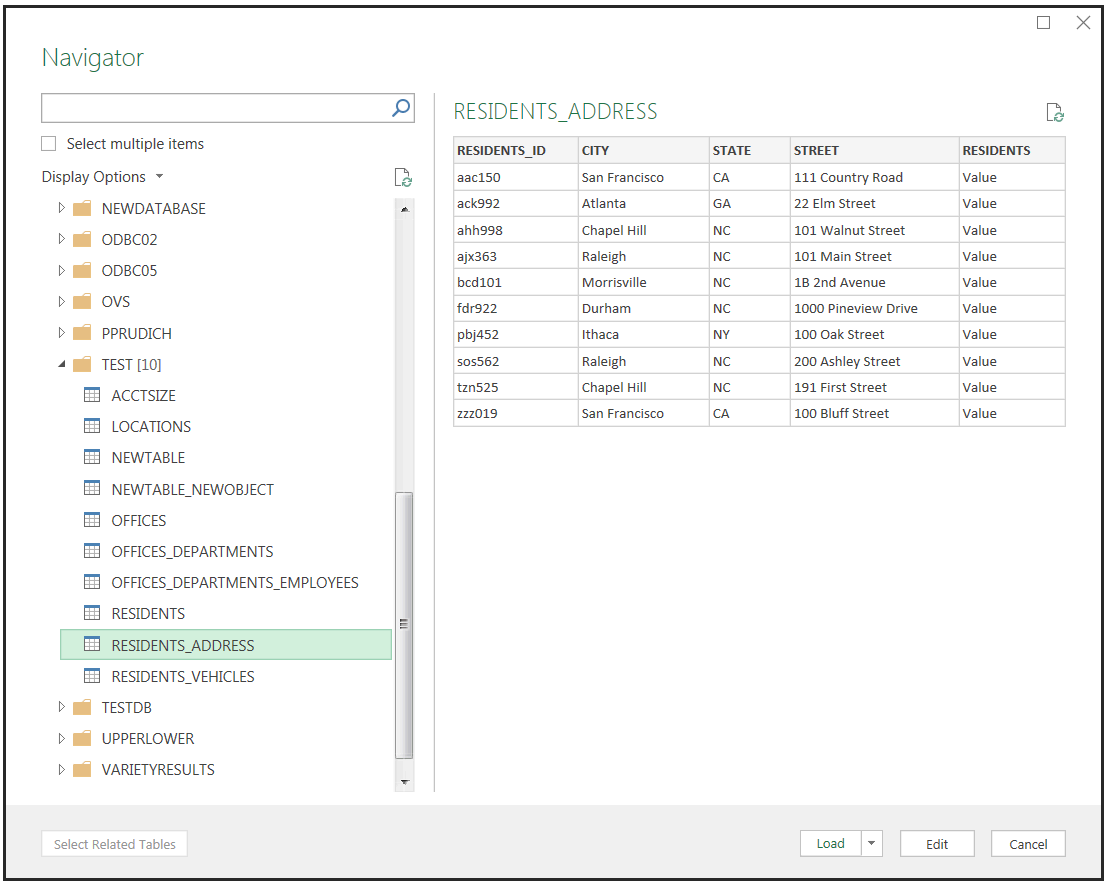
To use the driver to access data with Excel from the Data Connection Wizard:

1. Open your workbook in Excel.
2. From the **Data** menu, select **Get Data**>**From Other Sources**>**From ODBC**.
3. The **From ODBC** dialog appears.



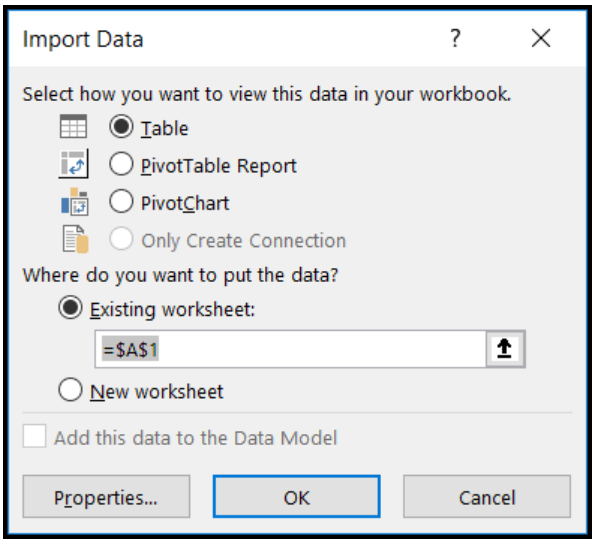
Select your data source from the Data Source Name (DSN) drop down; then, click **OK**.

4. You are prompted for logon credentials for your data source:
   - If your data source does not require logon credentials or if you prefer to specify your credentials using a connection string, select **Default or Custom** from the menu on the left. Optionally, specify your credential-related properties using a connection string in the provided field. Click **Connect** to proceed.
   - If your data source uses Windows credentials, select **Windows** from the menu; then, provide your credentials. Optionally, specify a connection string with credential-related properties in the provided field. Click **Connect** to proceed.
   - If your data source uses credentials stored on the database, select **Database**; then, provide your user name and password. Optionally, specify a connection string in the provided field. Click **Connect** to proceed.
5. The **Navigator** window appears.



From the list, select the tables you want to access. A preview of your data will appear in the pane on the right. Optionally, click **Edit** to modify the results using the Query Editor. Refer to the Microsoft Excel product documentation for detailed information on using the Query Editor.

6. Load your data:
   - Click **Load** to import your data into your work sheet. Skip to the end.
   - Click **Load>Load To** to specify a location to import your data. Proceed to the next step.
7. The **Import Data** window appears.



Select the desired view and insertion point for the data. Click **OK**.

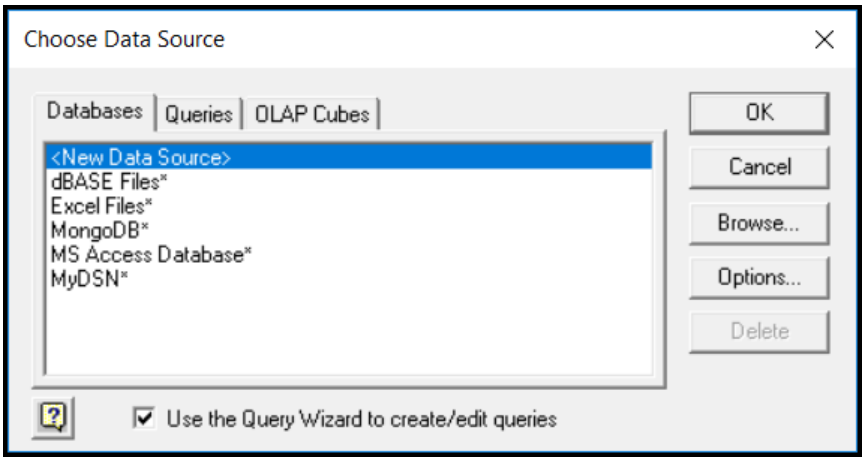
You have successfully accessed your data in Excel. For more information, refer to the Microsoft Excel product documentation at: https://support.office.com/.

Accessing data in Microsoft Excel from the Query Wizard (Windows only)

After you have configured your data source, you can use the driver to access your data with Microsoft Excel from the Query Wizard. Using the driver with Excel provides improved performance when retrieving data, while leveraging the driver's relational-mapping tools.

To use the driver to access data with Excel from the Query Wizard:

1. Open your workbook in Excel.
2. From the **Data** menu, select **Get Data**>**From Other Sources**>**From Microsoft Query**.
3. The **Choose Data Source** dialog appears.

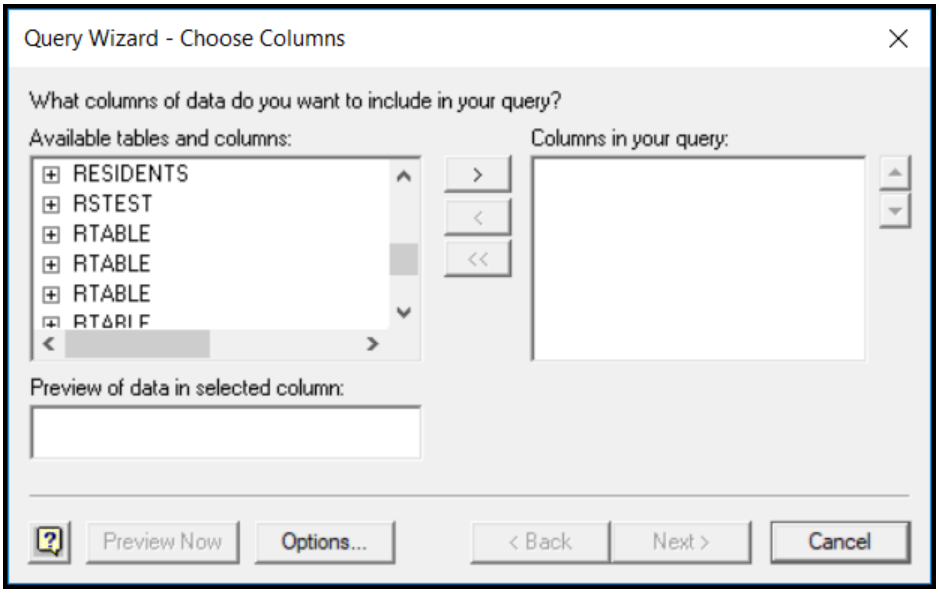


From the Databases list, select your data source. For example, **MyDSN**. Click **OK**.

4. The logon dialog appears pre-populated with the connection information you provided in your data source. If required, type your password. Click **OK** to proceed.

Note: The logon dialog may reappear if Excel needs to access additional information from the data source. If this occurs, re-enter your password; then, click **OK** to proceed to the next step.

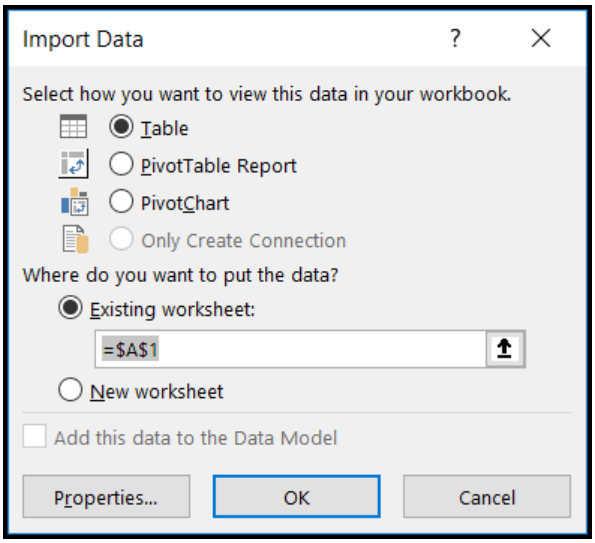
5. The **Query Wizard - Choose Columns** window appears.



Choose the columns you want to import into your workbook. To add a column, select the column name in Available tables and columns pane; then, click the **>** button. After you add the columns you want to include, click **Next** to continue.

6. Optionally, filter your data using the drop-down menus; then, click **Next**.
7. Optionally, sort your data using the drop-down menus; then, click **Next**.
8. Select "Return Data to Microsoft Excel"; then, click **Finish**.

9. The **Import Data** window appears.



Select the desired view and insertion point for your data. Click **OK**.

You have successfully accessed your data in Excel using the Query Wizard. For more information, refer to the Microsoft Excel product documentation at: https://support.office.com/.

4

Using the driver

This chapter guides you through the configuring and connecting to data sources. In addition, it explains how to use the functionality supported by your driver.

Related Links

For details, see the following topics:

- Configuring and connecting to data sources
- Using failover
- Using security
- Using DataDirect Connection Pooling
- Using DataDirect Bulk Load
- Using IP addresses
- XA interface support
- Binding parameter markers
- Isolation and lock levels supported
- Number of connections and statements supported
- SQL support
- Using arrays of parameters

- Support for Azure Synapse Analytics and Analytics Platform System
- Inserts on IDENTITY columns in data replication scenarios
- Packet logging

Configuring and connecting to data sources

After you install the driver, you configure data sources to connect to the database. See "Getting started" for an explanation of different types of data sources. The data source contains connection options that allow you to tune the driver for specific performance. If you want to use a data source but need to change some of its values, you can either modify the data source or override its values at connection time through a connection string.

If you choose to use a connection string, you must use specific connection string attributes. See "Connection option descriptions" for an alphabetical list of driver connection string attributes and their initial default values.

Configuring the product on UNIX/Linux

UNIX® This chapter contains specific information about using your driver in the UNIX and Linux environments.

See "Environment variables" for additional platform information.

Environment variables

The first step in setting up and configuring the driver for use is to set several environment variables. The following procedures require that you have the appropriate permissions to modify your environment and to read, write, and execute various files. You must log in as a user with full r/w/x permissions recursively on the entire Progress DataDirect *for* ODBC installation directory.

Related Links

For details, see the following topics:

- Library search path
- ODBCINI
- ODBCINST
- DD_INSTALLDIR

Library search path

The library search path variable can be set by executing the appropriate shell script located in the ODBC home directory. From your login shell, determine which shell you are running by executing:

```
echo $SHELL
```

C shell login (and related shell) users must execute the following command before attempting to use ODBC-enabled applications:

```
source ./odbc.csh
```

Bourne shell login (and related shell) users must initialize their environment as follows:

```
. ./odbc.sh
```

Executing these scripts sets the appropriate library search path environment variable:

- `LD_LIBRARY_PATH` on on HP-UX IPF, Linux, and Oracle Solaris
- `LIBPATH` on AIX
- `SHLIB_PATH` on HP-UX PA-RISC

The library search path environment variable must be set so that the ODBC core components and drivers can be located at the time of execution. After running the setup script, execute:

```
env
```

to verify that the *`installation_directory`*/lib directory has been added to your shared library path.

ODBCINI

Setup installs in the product installation directory a default system information file, named odbc.ini, that contains data sources. See "Data source configuration on UNIX/Linux" for an explanation of the odbc.ini file. The system administrator can choose to rename the file and/or move it to another location. In either case, the environment variable `ODBCINI` must be set to point to the fully qualified path name of the odbc.ini file.

For example, to point to the location of the file for an installation on /opt/odbc in the C shell, you would set this variable as follows:

```
setenv ODBCINI /opt/odbc/odbc.ini
```

In the Bourne or Korn shell, you would set it as:

```
ODBCINI=/opt/odbc/odbc.ini;export ODBCINI
```

As an alternative, you can choose to make the odbc.ini file a hidden file and not set the `ODBCINI` variable. In this case, you would need to rename the file to .odbc.ini (to make it a hidden file) and move it to the user's $HOME directory.

The driver searches for the location of the odbc.ini file as follows:

1. The driver checks the `ODBCINI` variable
2. The driver checks $HOME for .odbc.ini

If the driver does not locate the system information file, it returns an error.

ODBCINST

Setup installs in the product installation directory a default file, named odbcinst.ini, for use with DSN-less connections. See "DSN-less connections" for an explanation of the odbcinst.ini file. The system administrator can choose to rename the file or move it to another location. In either case, the environment variable `ODBCINST` must be set to point to the fully qualified path name of the odbcinst.ini file.

For example, to point to the location of the file for an installation on /opt/odbc in the C shell, you would set this variable as follows:

```
setenv ODBCINST /opt/odbc/odbcinst.ini
```

In the Bourne or Korn shell, you would set it as:

```
ODBCINST=/opt/odbc/odbcinst.ini;export ODBCINST
```

As an alternative, you can choose to make the odbcinst.ini file a hidden file and not set the `ODBCINST` variable. In this case, you would need to rename the file to .odbcinst.ini (to make it a hidden file) and move it to the user's `$HOME` directory.

The driver searches for the location of the odbcinst.ini file as follows:

1. The driver checks the `ODBCINST` variable
2. The driver checks `$HOME` for .odbcinst.ini

If the driver does not locate the odbcinst.ini file, it returns an error.

DD_INSTALLDIR

This variable provides the driver with the location of the product installation directory so that it can access support files. `DD_INSTALLDIR` must be set to point to the fully qualified path name of the installation directory.

For example, to point to the location of the directory for an installation on /opt/odbc in the C shell, you would set this variable as follows:

```
setenv DD_INSTALLDIR /opt/odbc
```

In the Bourne or Korn shell, you would set it as:

```
DD_INSTALLDIR=/opt/odbc;export DD_INSTALLDIR
```

The driver searches for the location of the installation directory as follows:

1. The driver checks the `DD_INSTALLDIR` variable
2. The driver checks the odbc.ini or the odbcinst.ini files for the InstallDir keyword (see "Configuration through the system information (odbc.ini) file" for a description of the `InstallDir` keyword)

If the driver does not locate the installation directory, it returns an error.

The next step is to test load the driver.

The test loading tool

The second step in preparing to use a driver is to test load it.

The ivtestlib (32-bit driver) and ddtestlib (64-bit driver) test loading tools are provided to test load drivers and help diagnose configuration problems in the UNIX and Linux environments, such as environment variables not correctly set or missing database client components. This tool is installed in the /bin subdirectory in the product installation directory. It attempts to load a specified ODBC driver and prints out all available error information if the load fails.

The test loading tool is provided to test load drivers and help diagnose configuration problems in the UNIX and Linux environments, such as environment variables not correctly set or missing database client components. This tool is installed in the bin subdirectory in the product installation directory. It attempts to load a specified ODBC driver and prints out all available error information if the load fails.

For example, if the drivers are installed in /opt/odbc/lib, the following command attempts to load the 32-bit driver on Solaris, where *xx* represents the version number of the driver:

```
ivtestlib /opt/odbc/lib/ivsqlsxx.so
```

Note: On Solaris, AIX, and Linux, the full path to the driver does not have to be specified for the tool. The HP-UX version, however, requires the full path.

If the load is successful, the tool returns a success message along with the version string of the driver. If the driver cannot be loaded, the tool returns an error message explaining why.

See "Version string information" for details about version strings.

The next step is to configure a data source through the system information file.

Data source configuration on UNIX/Linux

In the UNIX and Linux environments, a system information file is used to store data source information. Setup installs a default version of this file, called odbc.ini, in the product installation directory. This is a plain text file that contains data source definitions.

Related Links

For details, see the following topics:

- Configuration Through the System Information (odbc.ini) File

Configuration Through the System Information (odbc.ini) File

To configure a data source manually, you edit the odbc.ini file with a text editor. The content of this file is divided into three sections.

Note: The driver and driver manager support ASCII and UTF-8 encoding in the odbc.ini file.

Refer to the "Character encoding in the odbc.ini and odbcinst.ini files" in *Progress DataDirect for ODBC*

Drivers Reference for details.

At the beginning of the file is a section named `[ODBC Data Sources]` containing *`data_source_name=installed-driver`* pairs, for example:

```
SQL Server=DataDirect 8.0 SQL Server Wire Protocol Driver
.
```

The driver uses this section to match a data source to the appropriate installed driver.

The `[ODBC Data Sources]` section also includes data source definitions. The default odbc.ini contains a data source definition for the driver. Each data source definition begins with a data source name in square brackets, for example, `[SQL Server]`. The data source definitions contain connection string *attribute=value* pairs with default values. You can modify these values as appropriate for your system. "Connection option descriptions" describes these attributes. See "Sample default odbc.ini file" for sample data sources.

The second section of the file is named `[ODBC File DSN]` and includes one keyword:

```
[ODBC File DSN]
DefaultDSNDir=
```

This keyword defines the path of the default location for file data sources (see "File data sources").

Note: This section is not included in the default odbc.ini file that is installed by the product installer. You must add this section manually.

The third section of the file is named `[ODBC]` and includes several keywords, for example:

```
[ODBC]
IANAAppCodePage=4
InstallDir=/opt/odbc
Trace=0
TraceFile=odbctrace.out
TraceDll=/opt/odbc/lib/ivtrc28.so
ODBCTraceMaxFileSize=102400
ODBCTraceMaxNumFiles=10
```

The IANAAppCodePage keyword defines the default value that the UNIX/Linux driver uses if individual data sources have not specified a different value. See "IANAAppCodePage" in "Connection option descriptions" for details. The default value is `4`.

For supported code page values, refer to "Code page values" in the *Progress DataDirect for ODBC Drivers Reference*.

The `InstallDir` keyword must be included in this section. The value of this keyword is the path to the installation directory under which the /lib and /locale directories are contained. The installation process automatically writes your installation directory to the default odbc.ini file.

For example, if you choose an installation location of /opt/odbc, then the following line is written to the

[ODBC] section of the default odbc.ini:

```
InstallDir=/opt/odbc
```

Note: If you are using only DSN-less connections through an odbcinst.ini file and do not have an odbc.ini file, then you must provide [ODBC] section information in the [ODBC] section of the odbcinst.ini file. The driver and Driver Manager always check first in the [ODBC] section of an odbc.ini file. If no odbc.ini file exists or if the odbc.ini file does not contain an [ODBC] section, they check for an [ODBC] section in the odbcinst.ini file. See "DSN-less connections" for details.

ODBC tracing allows you to trace calls to the ODBC driver and create a log of the traces for troubleshooting purposes. The following keywords all control tracing: Trace, TraceFile, TraceDLL, ODBCTraceMaxFileSize, and ODBCTraceMaxNumFiles.

For a complete discussion of tracing, refer to "ODBC trace" in the *Progress DataDirect for ODBC Drivers Reference*.

Related Links

For details, see the following topics:

- Sample default odbc.ini file

The following is a sample odbc.ini file that Setup installs in the installation directory. All occurrences of ODBCHOME are replaced with your installation directory path during installation of the file. Values that you must supply are enclosed by angle brackets (< >). If you are using the installed odbc.ini file, you must supply the values and remove the angle brackets before that data source section will operate properly. Commented lines are denoted by the # symbol. This sample shows a 32-bit driver with the driver file name beginning with iv. A 64-bit driver file would be identical except that driver name would begin with dd and the list of data sources would include only the 64-bit drivers.

```
[ODBC Data Sources]
SQL Server=DataDirect 8.0 SQL Server Wire Protocol

[SQL Server]
Driver=ODBCHOME/lib/ivsqls28.so
Description=DataDirect 8.0 SQL Server Wire Protocol
AEKeyCacheTTL=-1
AEKeystoreClientSecret=
AEKeyStoreLocation=
AEKeystorePrincipalId=
AEKeyStoreSecret=
AlternateServers=
AlwaysReportTriggerResults=0
AnsiNPW=1
ApplicationIntent=0
ApplicationName=
ApplicationUsingThreads=1
AuthenticationMethod=1
BulkBinaryThreshold=32
BulkCharacterThreshold=-1
BulkLoadBatchSize=1024
```

```
BulkLoadFieldDelimiter=
BulkLoadOptions=2
BulkLoadThreshold=2
BulkLoadRecordDelimiter=
ColumnEncryption=Disabled
ConnectionReset=0
ConnectionRetryCount=0
ConnectionRetryDelay=3
CryptoLibName=
CryptoProtocolVersion=TLSv1.2,TLSv1.1,TLSv1
Database=<database_name>
DefaultLongDataBuffLen=1024
EnableBulkLoad=0
EnableFIPS=1
EnableQuotedIdentifiers=0
EnableServersideCursors=1
EncryptionMethod=0
FailoverGranularity=0
FailoverMode=0
FailoverPreconnect=0
FetchTSWTZasTimestamp=0
FetchTWFSasTime=1
GSSClient=native
HostName=<SQL_Server_host>
HostNameInCertificate=
InitializationString=
KeepAlive=0
KeepConnectionActive=0
Language=
LoadBalanceTimeout=0
LoadBalancing=0
LoginTimeout=15
LogonID=
MaxPoolSize=100
MinPoolSize=0
MultiSubnetFailover=0
PacketSize=-1
Password=
Pooling=0
PortNumber=<SQL_Server_server_port>
PRNGSeedFile=/dev/random
PRNGSeedSource=0
ProxyHost=
ProxyMode=0
ProxyPassword=
ProxyPort=0
ProxyUser=
QueryTimeout=0
ReportCodePageConversionErrors=0
SnapshotSerializable=0
SocketIdleTimeCheckInterval=1500
SSLLibName=
TrustStore=
TrustStorePassword=
```

```
ValidateServerCertificate=1
WorkStationID=
XMLDescribeType=-10

[ODBC]
IANAAppCodePage=4
InstallDir=ODBCHOME
Trace=0
TraceFile=odbctrace.out
TraceDll=ODBCHOME/lib/ivtrc28.so
ODBCTraceMaxFileSize=102400
ODBCTraceMaxNumFiles=10
[ODBC File DSN]
DefaultDSNDir=
UseCursorLib=0
```

To modify or create data sources in the odbc.ini file, use the following procedures.

- **To modify a data source:**
  1. Using a text editor, open the odbc.ini file.
  2. Modify the default attributes in the data source definitions as necessary based on your system specifics, for example, enter the host name and port number of your system in the appropriate location.

     Consult the "SQL Server Wire Protocol Attribute Names" table in the "Connection options descriptions" for other specific attribute values.

  3. After making all modifications, save the odbc.ini file and close the text editor.

     Important: The "Connection option descriptions" section lists both the long and short names of the attribute. When entering attribute names into odbc.ini, you must use the long name of the attribute. The short name is not valid in the odbc.ini file.

- **To create a new data source:**
  1. Using a text editor, open the odbc.ini file.
  2. Copy an appropriate existing default data source definition and paste it to another location in the file.
  3. Change the data source name in the copied data source definition to a new name. The data source name is between square brackets at the beginning of the definition, for example, `[SQL Server]`.
  4. Modify the attributes in the new definition as necessary based on your system specifics, for example, enter the host name and port number of your system in the appropriate location.

     Consult the "SQL Server Wire Protocol Attribute Names" table in the "Connection option descriptions" for other specific attribute values.

  5. In the `[ODBC]` section at the beginning of the file, add a new *data_source_name=installed-driver* pair containing the new data source name and the appropriate installed driver name.
  6. After making all modifications, save the odbc.ini file and close the text editor.

     Important: The "SQL Server Wire Protocol Attribute Names" table in the "Connection option descriptions" section lists both the long and short name of the attribute. When entering attribute names into odbc.ini, you must use the long name of the attribute. The short name is not valid in the odbc.ini file.

The example application

Progress DataDirect ships an application, named *example*, that is installed in the /samples/example subdirectory of the product installation directory. Once you have configured your environment and data source, use the example application to test passing SQL statements. To run the application, enter `example` and follow the prompts to enter your data source name, user name, and password.

If successful, a SQL> prompt appears and you can type in SQL statements, such as `SELECT * FROM table_name`. If *example* is unable to connect to the database, an appropriate error message appears.

Refer to the example.txt file in the example subdirectory for an explanation of how to build and use this application.

Refer to "The example application" in *Progress DataDirect for ODBC Drivers Reference* for more information.

DSN-less connections

Connections to a data source can be made via a connection string without referring to a data source name (DSN-less connections). This is done by specifying the `DRIVER=` keyword instead of the `DSN=` keyword in a connection string, as outlined in the ODBC specification. A file named odbcinst.ini must exist when the driver encounters `DRIVER=` in a connection string.

Setup installs a default version of this file in the product installation directory (see "ODBCINST" for details about relocating and renaming this file). This is a plain text file that contains default DSN-less connection information. You should not normally need to edit this file. The content of this file is divided into several sections.

At the beginning of the file is a section named `[ODBC Drivers]` that lists installed drivers, for example,

```
DataDirect 8.0 SQL Server Wire Protocol=Installed.
```

This section also includes additional information for each driver.

The next section of the file is named `[Administrator]`. The keyword in this section, `AdminHelpRootDirectory`, is required for the Linux ODBC Administrator to locate its help system. The installation process automatically provides the correct value for this keyword.

The final section of the file is named `[ODBC]`. The `[ODBC]` section in the odbcinst.ini file fulfills the same purpose in DSN-less connections as the `[ODBC]` section in the odbc.ini file does for data source connections. See "Configuration through the system information (odbc.ini) file" for a description of the other keywords this section.

Note: The odbcinst.ini file and the odbc.ini file include an `[ODBC]` section. If the information in these two sections is not the same, the values in the odbc.ini `[ODBC]` section override those of the odbcinst.ini `[ODBC]` section.

Related Links

For details, see the following topics:

- Sample odbcinst.ini File

Sample odbcinst.ini File

The following is a sample odbcinst.ini. All occurrences of ODBCHOME are replaced with your installation directory path during installation of the file. Commented lines are denoted by the # symbol. This sample shows a 32-bit driver with the driver file name beginning with iv; a 64-bit driver file would be identical except that driver names would begin with dd.

```
[ODBC Drivers]
DataDirect 8.0 SQL Server Wire Protocol=Installed

[DataDirect 8.0 SQL Server Wire Protocol]
Driver=ODBCHOME/lib/ivsqls28.so
APILevel=1
ConnectFunctions=YYY
DriverODBCVer=3.52
FileUsage=0
HelpRootDirectory=ODBCHOME/SQLServerHelp
Setup=ODBCHOME/lib/ivsqls28.so
SQLLevel=1

[ODBC]
#This section must contain values for DSN-less connections
#if no odbc.ini file exists. If an odbc.ini file exists,
#the values from that [ODBC] section are used.

IANAAppCodePage=4
InstallDir=ODBCHOME
Trace=0
TraceFile=odbctrace.out
TraceDll=ODBCHOME/lib/ivtrc28.so
ODBCTraceMaxFileSize=102400
ODBCTraceMaxNumFiles=10
```

File data sources

The Driver Manager on UNIX and Linux supports file data sources. The advantage of a file data source is that it can be stored on a server and accessed by other machines, either Windows, UNIX, or Linux. See "Getting started" for a general description of ODBC data sources on both Windows and UNIX.

A file data source is simply a text file that contains connection information. It can be created with a text editor. The file normally has an extension of .dsn.

For example, a file data source for the driver would be similar to the following:

```
[ODBC]
Driver=DataDirect 8.0 SQL Server Wire Protocol
Database=default
Port=1433
HostName=SQLServer2
LogonID=JOHN
```

It must contain all basic connection information plus any optional attributes. Because it uses the "DRIVER=" keyword, an odbcinst.ini file containing the driver location must exist (see "DSN-less connections").

The file data source is accessed by specifying the `FILEDSN=` instead of the `DSN=` keyword in a connection string, as outlined in the ODBC specification. The complete path to the file data source can be specified in the syntax that is normal for the machine on which the file is located. For example, on Windows:

```
FILEDSN=C:\Program Files\Common Files\ODBC\DataSources\SQLServer2.dsn
```

or, on UNIX and Linux:

```
FILEDSN=/home/users/john/filedsn/SQLServer2.dsn
```

If no path is specified for the file data source, the Driver Manager uses the DefaultDSNDir property, which is defined in the `[ODBC File DSN]` setting in the odbc.ini file to locate file data sources (see "Data source configuration on UNIX/Linux" for details). If the `[ODBC File DSN]` setting is not defined, the Driver Manager uses the InstallDir setting in the [ODBC] section of the odbc.ini file. The Driver Manager does not support the SQLReadFileDSN and SQLWriteFileDSN functions.

As with any connection string, you can specify attributes to override the default values in the data source:

```
FILEDSN=/home/users/john/filedsn/SQLServer2.dsn;UID=james;PWD=test01
```

UTF-16 applications on UNIX and Linux

Because the DataDirect Driver Manager allows applications to use either UTF-8 or UTF-16 Unicode encoding, applications written in UTF-16 for Windows platforms can also be used on UNIX and Linux platforms.

The Driver Manager assumes a default of UTF-8 applications; therefore, two things must occur for it to determine that the application is UTF-16:

- The definition of SQLWCHAR in the ODBC header files must be switched from "char *" to "short *". To do this, the application uses #define SQLWCHARSHORT.
- The application must set the encoding for the environment or connection using one of the following attributes. If your application passes UTF-8 encoded strings to some connections and UTF-16 encoded strings to other connections in the same environment, encoding should be set for the connection only; otherwise, either method can be used.
  - To configure the encoding for the environment, set the ODBC environment attribute SQL_ATTR_APP_UNICODE_TYPE to a value of `SQL_DD_CP_UTF16`, for example:

    ```
    rc = SQLSetEnvAttr(*henv,
    SQL_ATTR_APP_UNICODE_TYPE,(SQLPOINTER)SQL_DD_CP_UTF16, SQL_IS_INTEGER);
    ```

  - To configure the encoding for the connection only, set the ODBC connection attribute SQL_ATTR_APP_UNICODE_TYPE to a value of SQL_DD_CP_UTF16. For example:

    ```
    rc = SQLSetConnectAttr(hdbc, SQL_ATTR_APP_UNICODE_TYPE, SQL_DD_CP_UTF16,
    SQL_IS_INTEGER);
    ```

Data source configuration through a GUI

On Windows, data sources are stored in the Windows Registry. You can configure and modify data

sources through the ODBC Administrator using a driver Setup dialog box, as described in this section.

When the driver is first installed, the values of its connection options are set by default. These values appear on the driver Setup dialog box tabs when you create a new data source. You can change these default values by modifying the data source. In the following procedure, the description of each tab is followed by a table that lists the connection options for that tab and their initial default values. This table links you to a complete description of the options and their connection string attribute equivalents. The connection string attributes are used to override the default values of the data source if you want to change these values at connection time.

To configure a SQL Server data source:

1. Start the ODBC Administrator by selecting its icon from the Progress DataDirect for ODBC program group.
2. Select a tab:
   - **User DSN**: If you are configuring an existing user data source, select the data source name and click **Configure** to display the driver Setup dialog box.

     If you are configuring a new user data source, click **Add** to display a list of installed drivers. Select the driver and click **Finish** to display the driver Setup dialog box.

   - **System DSN**: If you are configuring an existing system data source, select the data source name and click **Configure** to display the driver Setup dialog box.

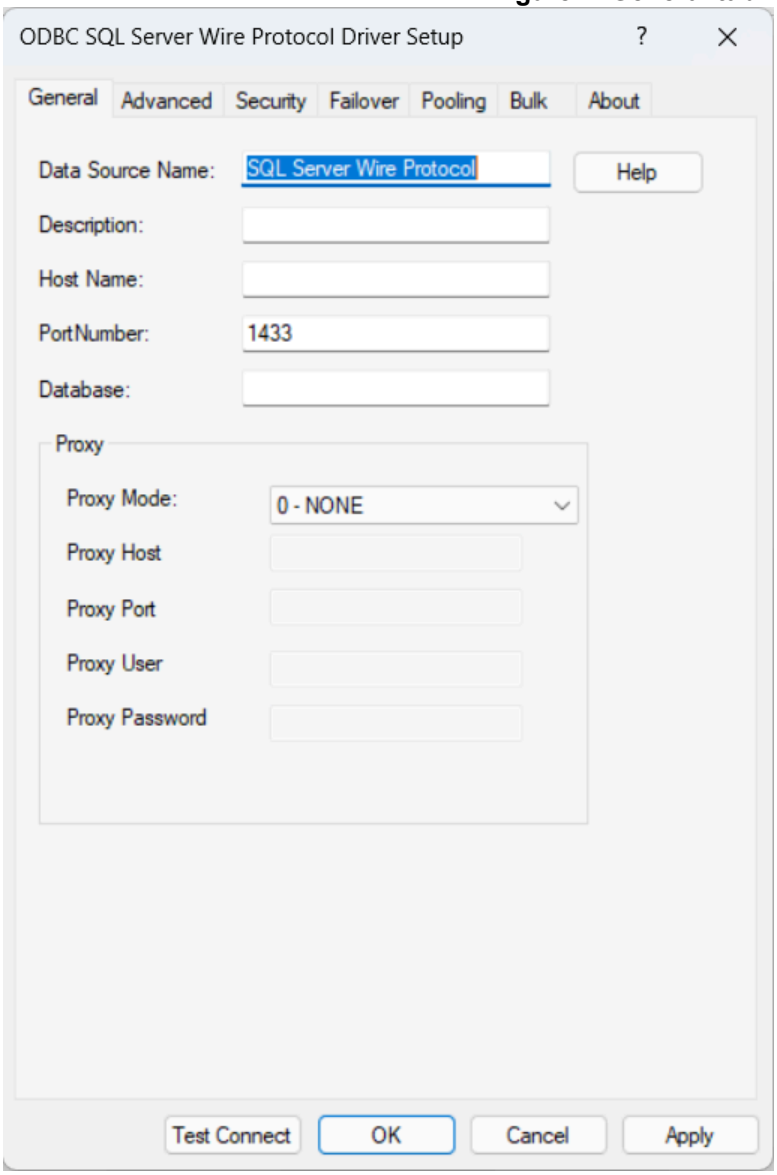
     If you are configuring a new system data source, click **Add** to display a list of installed drivers. Select the driver and click **Finish** to display the driver Setup dialog box.

   - **File DSN**: If you are configuring an existing file data source, select the data source file and click **Configure** to display the driver Setup dialog box.

     If you are configuring a new file data source, click **Add** to display a list of installed drivers; then, select a driver. Click **Advanced** if you want to specify attributes; otherwise, click **Next** to proceed. Specify a name for the data source and click **Next**. Verify the data source information; then, click **Finish** to display the driver Setup dialog box.

3. The General tab of the Setup dialog box appears by default.

Figure 1. General tab



On this tab, provide values for the options in the following table; then, click **Apply.** The table provides links to descriptions of the connection options. The General tab displays fields that are required for creating a data source. The fields on all other tabs are optional, unless noted otherwise.

Connection Options: General	Description
Data Source Name	Specifies the name of a data source in your Windows Registry or odbc.ini file. **Default:** None
Description	Specifies an optional long description of a data source. This description is not used as a runtime connection attribute, but does appear in the

Connection Options: General	Description
	`ODBC.INI` section of the Registry and in the odbc.ini file. **Default:** None
Host Name	The server to which you want to connect. This value can be a: • IP address • named server • named instance • server name • virtual network name See "Host Name" for details. **Default:** None
Port Number	Specifies the port number of the server listener. **Default:** `1433`
Database	Specifies the name of the database to which you want to connect. **Default:** None
Proxy Mode	Determines whether the driver connects to your data source endpoint through an HTTP proxy server. If set to **0 - NONE**, the driver connects directly to the data source endpoint specified by the Host Name connection option. If set to **1 - HTTP**, the driver connects to the data source endpoint through the HTTP proxy server specified by the ProxyHost connection option. Default: **0 - None**
Proxy Host	Specifies the Hostname and possibly the Domain of the Proxy Server. The value specified can be a host name, a fully qualified domain name, or an IPv4 or IPv6 address. **Default:** Empty string
Proxy Port	Specifies the port number where the Proxy Server is listening for HTTP requests. Default: `0`
Proxy User	Specifies the user name needed to connect to the Proxy Server.

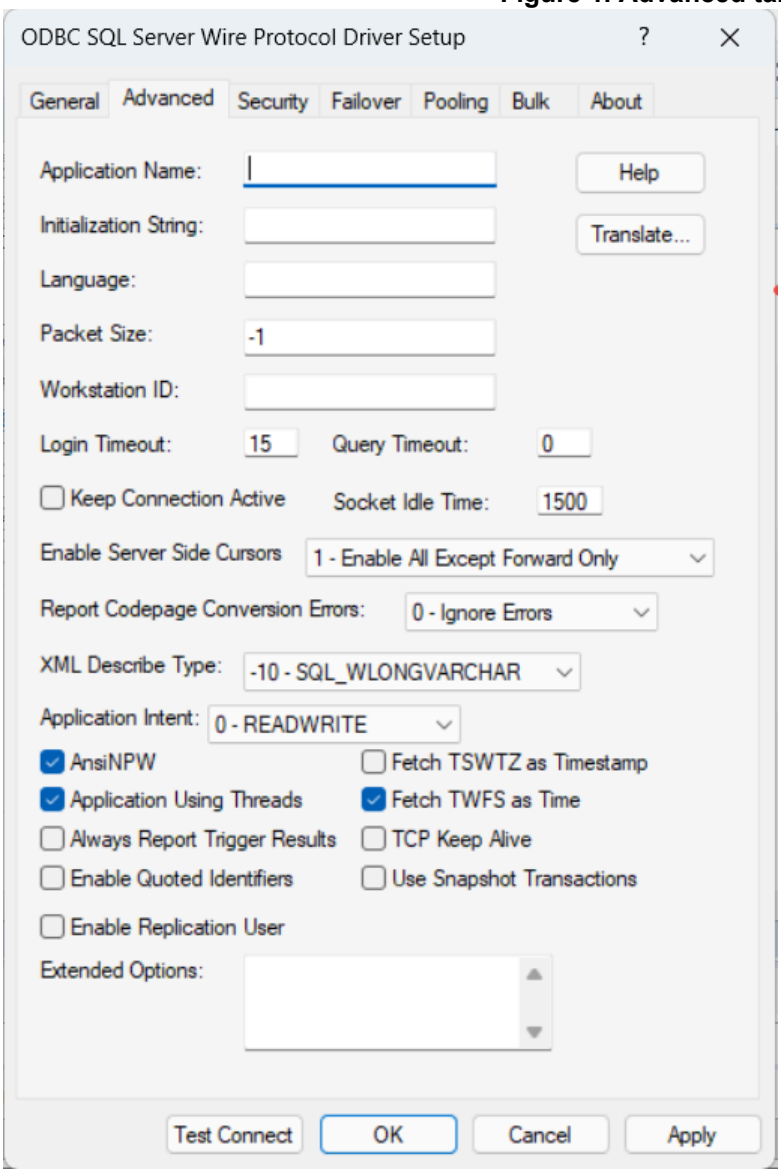
Connection Options: General	Description
	Default: Empty string
Proxy Password	Specifies the password needed to connect to the Proxy Server. **Default:** Empty string

4. At any point during the configuration process, you can click **Test Connect** to attempt to connect to the data source using the connection options specified in the driver Setup dialog box. A logon dialog box appears (see "Using a logon dialog box" for details). Note that the information you enter in the logon dialog box during a test connect is not saved.
5. To further configure your driver, click on the following tabs. The corresponding sections provide details on the fields specific to each configuration tab:
   - Advanced tab allows you to configure advanced behavior.
   - Security tab allows you to specify security data source settings.
   - Failover tab allows you to specify failover data source settings.
   - Pooling tab allows you to specify connection pooling settings.
   - Bulk tab allows you to specify data source settings for DataDirect Bulk Load.
6. Click **OK**. When you click **OK**, the values you have specified become the defaults when you connect to the data source. You can change these defaults by using this procedure to reconfigure your data source. You can override these defaults by connecting to the data source using a connection string with alternate values.

Advanced tab

The Advanced tab allows you to specify additional data source settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

Figure 1. Advanced tab



Connection Options: Advanced	Description
Application Name	The name the database uses to identify your application. **Default:** None
Initialization String	A SQL command that is issued immediately after connecting to the database to manage session settings. **Default:** None
Language	The national language to use for Microsoft SQL Server system messages. If no

Connection Options: Advanced	Description
	language is specified, the connection uses the default language specified for the login on the server. **Default:** None.
Packet Size	Determines the number of bytes for each database protocol packet that is transferred from the database server to the client machine. If set to `-1`, the driver uses the maximum packet size that is set by the database server. If set to `0`, the driver uses the default packet size that is used by the database server. If set to *x*, an integer from `1` to `127`, the driver uses a packet size that is a multiple of 512 bytes. For example, `PacketSize=8` means to set the packet size to 8 * 512 bytes (4096 bytes). **Default:** `-1`
Workstation ID	The workstation ID that is used by the client. **Default:** None
Login Timeout	The number of seconds the driver waits for a connection to be established before returning control to the application and generating a timeout error. If set to `-1`, the connection request does not time out. The driver silently ignores the SQL_ATTR_LOGIN_TIMEOUT attribute. If set to `0`, the connection request does not time out, but the driver responds to the SQL_ATTR_LOGIN_TIMEOUT attribute. If set to *x*, the connection request times out after the specified number of seconds unless the application overrides this setting with the SQL_ATTR_LOGIN_TIMEOUT attribute. **Default:** `15`
Query Timeout	The number of seconds for the default query timeout for all statements that are created by a connection. If set to `-1`, the query does not time out. The driver silently ignores the SQL_ATTR_QUERY_TIMEOUT attribute. If set to `0`, the query does not time out, but the driver responds to the SQL_ATTR_QUERY_TIMEOUT attribute. If set to *x*, all queries time out after the specified number of seconds unless the

Connection Options: Advanced	Description
	application overrides this value by setting the SQL_ATTR_QUERY_TIMEOUT attribute. **Default:** `0`
Keep Connection Active	Specifies whether the driver periodically sends lightweight SQL operations to the database after a connection has been idle for the time specified by the Socket Idle Time (`SocketIdleTimeCheckInterval`) option. The SQL operation resets the Azure SQL Gateway or database idle timeout timer to keep the connection open. If disabled, the driver does *not* send lightweight SQL operations the database to keep the connection open. Once a connection is idle for the duration specified by the Azure SQL Gateway or database, the connection times out. If enabled, the driver periodically sends lightweight SQL operations to the database to keep the connection active. Once a connection is idle for the duration specified by the Socket Idle Time option, the driver executes a lightweight query (`Select 0`) to the database to prevent the connection from timing out. **Default:** Disabled
Socket Idle Time	Specifies the interval of time, in seconds, at which the driver checks the connection for activity when Keep Connection Active is enabled (`KeepConnectionActive=1`). If no activity has been detected during this period, the driver issues a lightweight query (`Select 0`) to the database to maintain the connection. **Default:** `1500`
Enable Server Side Cursors	Determines which server-side cursors are enabled for the data source. This option applies to both Keyset and Static cursors. If set to **0 - Disabled**, all server-side scrollable cursors are disabled for the data source. Forward-only cursors on the server-side are enabled. If set to **1 - Enable All Except Forward Only**, all server-side scrollable cursors are enabled for the data source, while forward-only cursors on the server side are disabled. If set to **2 - Enable Forward Only for Rowset Size >1**, only forward-only cursors are enabled on the server-side when the rowset size set to a value greater than one. If set to **3 - Enable All**, all server-side cursors, scrollable and forward-only, are enabled for the data source. If set to **4 - Enable Forward Only for Select For Update**, forward-only cursors on the server-side are enabled only for Select For Update statements. For other Select statements, the driver uses forward-only cursors on the client-side. This setting avoids using driver emulation for other Select statements, thereby improving

Connection Options: Advanced	Description
	performance and allowing the use of native updatable result sets. **Default: 1 - Enable All Except Forward Only**
Report Codepage Conversion Errors	Specifies how the driver handles code page conversion errors that occur when a character cannot be converted from one character set to another. If set to **0 - Ignore Errors**, the driver substitutes 0x1A for each character that cannot be converted and does not return a warning or error. If set to **1 - Return Error**, the driver returns an error instead of substituting 0x1A for unconverted characters. If set to **2 - Return Warning**, the driver substitutes 0x1A for each character that cannot be converted and returns a warning. **Default: 0 - Ignore Errors**
XML Describe Type	The SQL data type that is returned by SQLGetTypeInfo for the XML data type. See Using the XML data type for further information about the XML data type. If set to **-4 - SQL_LONGVARBINARY**, the driver uses the description SQL_LONGVARBINARY for columns that are defined as the XML data type. If set to **-10 - SQL_WLONGVARCHAR**, the driver uses the description SQL_WLONGVARCHAR for columns that are defined as the XML data type. **Default: -10 - SQL_WLONGVARCHAR**
Application Intent	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 Always On Availability Groups have been deployed. If set to **0 - READWRITE**, the driver connects to a read-write node in the Always On environment. If set to **1 - READONLY**, the driver requests read-only routing and connects to the read-only database replicas specified by the server. **Default: 0 - READWRITE**
AnsiNPW	Determines whether ANSI-defined behaviors are exposed. Setting this option has no effect on NULL concatenation for Windows Azure SQL Database or SQL Server versions higher than SQL Server 2012. When enabled, the driver sets four ANSI-defined behaviors for handling NULL

Connection Options: Advanced	Description
	comparisons: NULLS, character data padding, warnings, and NULL concatenation. When disabled, ANSI-defined behaviors are not exposed. If the driver appears to be truncating trailing blank spaces, disable this attribute. **Default:** Enabled
Fetch TSWTZ as Timestamp	Determines whether the driver returns column values with the timestamp with time zone data type as the ODBC data type SQL_TYPE_TIMESTAMP or SQL_VARCHAR. If enabled, the driver returns column values with the timestamp with time zone data type as the ODBC type SQL_TYPE_TIMESTAMP. The time zone information in the fetched value is truncated. Use this value if your application needs to process values the same way as TIMESTAMP columns. If disabled, the driver returns column values with the timestamp with time zone data type as the ODBC data type SQL_VARCHAR. Use this value if your application requires the time zone information in the fetched value. **Default:** Disabled.
Application Using Threads	Determines whether the driver works with applications using multiple ODBC threads. If enabled, the driver works with single-threaded and multi-threaded applications. If disabled, the driver does not work with multi-threaded applications. If using the driver with single-threaded applications, this value avoids additional processing required for ODBC thread-safety standards. **Default:** Enabled
Fetch TWFS as Time	Determines whether the driver returns column values with the time data type as the ODBC data type SQL_TYPE_TIME or SQL_TYPE_TIMESTAMP. Supported only for Microsoft SQL Server 2008. If enabled, the driver returns column values with the time data type as the ODBC data type SQL_TYPE_TIME. The fractional seconds portion of the value is truncated. If disabled, the driver returns column values with the time data type as the ODBC data type SQL_TYPE_TIMESTAMP. The fractional seconds portion of the value is preserved. Time columns are not searchable when they are described and fetched as timestamp.
Always Report Trigger Results	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). For Microsoft SQL Server 2005 and higher and Windows Azure SQL

Connection Options: Advanced	Description
	Database, this includes triggers that are fired by Data Definition Language (DDL) events. If set to enabled, the driver returns all results, including results that are generated by triggers. Multiple trigger results are returned one at a time. You can use the SQLMoreResults function to return individual trigger results. Warnings and errors are reported in the results as they are encountered. If disabled: • For Microsoft SQL Server 2005 and higher and Windows Azure SQL Database, the driver does not report trigger results if the statement is a single INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, GRANT, REVOKE, or DENY statement. • For other Microsoft SQL Server databases, the driver does not report trigger results if the statement is a single INSERT, UPDATE, or DELETE statement. **Default:** Disabled
TCP Keep Alive	Specifies whether the driver enables TCPKeepAlive. If disabled, the driver does not enable TCPKeepAlive. If enabled, the driver enables TCPKeepAlive. **Default:** Disabled

Enable Quoted Identifiers	If enabled, the database enforces ANSI rules regarding quotation marks. Double quotation marks can only be used for identifiers, such as column and table names. Character strings must be enclosed in single quotation marks, for example:
	
	`SELECT "au_id"`
	`FROM "authors"`
	`WHERE "au_lname" = 'O''Brien'`
	
	If disabled, applications that use quoted identifiers encounter errors when they generate SQL statements with quoted identifiers.

Use Snapshot Transactions	Allows your application to use the snapshot isolation level if your Microsoft SQL Server database is configured for Snapshot isolation. Supported only for Microsoft SQL Server 2005 and higher. See Using the Snapshot isolation level for details about using the snapshot isolation level. When enabled and your application has the transaction isolation level set to serializable, the application uses the snapshot isolation level.

Connection Options: Advanced	Description
	When disabled and your application has the transaction isolation level set to serializable, the application uses the serializable isolation level. **Default:** Disabled
Enable Replication User	Specifies whether explicit values may be inserted into IDENTITY columns defined as NOT FOR REPLICATION. If enabled, the driver allows explicit inserts on IDENTITY columns defined as NOT FOR REPLICATION. If disabled, the driver enforces constraints on IDENTITY columns imposed by the NOT FOR REPLICATION flag. **Default:** Disabled.

Extended Options: Type a semi-colon separated list of connection options and their values. Use this configuration option to set the value of undocumented connection options that are provided by Progress DataDirect Customer Support. You can include any valid connection option in the Extended Options string, for example:

```
Database=myDB;UndocumentedOption1=value [;UndocumentedOption2=value;]
```

If the Extended Options string contains option values that are also set in the setup dialog or data source, the values of the options specified in the Extended Options string take precedence. However, connection options that are specified on a connection string override any option value specified in the Extended Options string.

If you finished configuring your driver, proceed to Step 6 in "Data source configuration through a GUI". Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

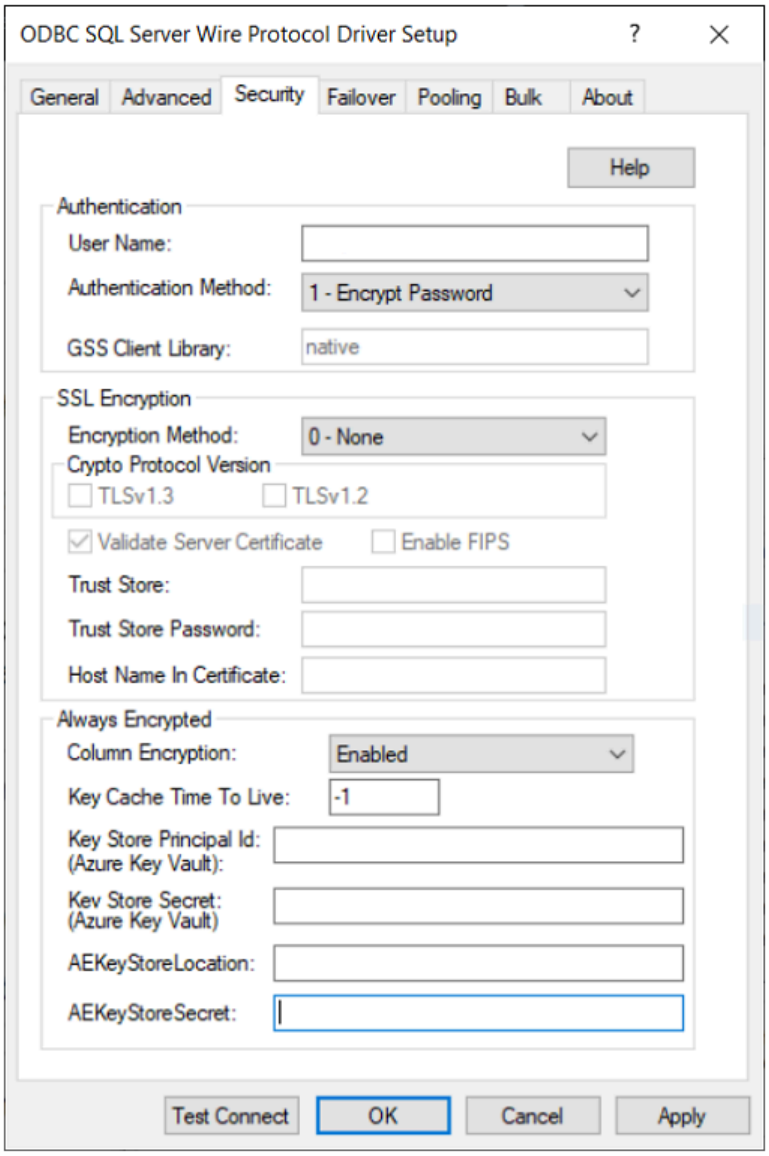
- General tab allows you to configure options that are required for creating a data source.
- Security tab allows you to specify security data source settings.
- Failover tab allows you to specify failover data source settings.
- Pooling tab allows you to specify connection pooling settings.
- Bulk tab allows you to specify data source settings for DataDirect Bulk Load.

Security tab

The Security Tab allows you to specify your security settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

See "Using security" for a general description of authentication and encryption and their configuration requirements.

Figure 1. Security tab



Connection Options: Security	Description
User Name	Specifies one of the following identifiers used for authentication: • The default user ID that is used to connect to your database. • The object (principal) ID of the Azure SQL logical server. This value is used when authenticating the service principal user with Entra ID (`AuthenticationMethod=36`). • The client ID for the user-assigned managed identity used for authentication. This is used when authenticating with Managed Identity authentication (`AuthenticationMethod=37`).

Connection Options: Security	Description
	Default: None
Authentication Method	Specifies the method the driver uses to authenticate the user to the server when a connection is established. If set to **1 - Encrypt Password**, the driver sends the user ID in clear text and an encrypted password to the server for authentication. If set to **4 - Kerberos Authentication**, the driver uses Kerberos authentication. This method supports both Microsoft Entra Kerberos and MIT Kerberos environments. Setting this value to **4 - Kerberos Authentication** also enables NTLMv2 and NTLMv1 authentication on Windows platforms. The protocol used for a connection is determined by the local security policy settings for the client. If set to **13 - Active Directory Password**, the driver authenticates using an Entra ID user name and password when connecting to a Azure SQL Database data store. All communications between the service are encrypted using TLS/SSL. If set to **36 - Active Directory Service Principal**, the driver authenticates using an Entra ID service principal when establishing a connection to an Azure SQL Database data store. All communications between the service are encrypted using TLS/SSL. If set to **37 - Active Directory Manged Identity**, the driver authenticates using a managed identity when accessing Azure resources. All communications between the service are encrypted using TLS/SSL. **Default: 1 - Encrypt Password**
GSS Client Library	The name of the GSS client library that the driver uses to communicate with the Key Distribution Center (KDC). **Default:** `native` (the driver uses the GSS client for Windows Kerberos.)
Encryption Method	The method the driver uses to encrypt data sent between the driver and the database server. If set to **0 - None**, data is not encrypted. If set to **1 - SSL**, data is encrypted using the TLS/SSL protocols specified in the Crypto Protocol Version connection option. If set to **6 - RequestSSL**, the login request and data are encrypted using TLS/SSL if the server is configured for TLS/SSL. If the server is not configured for TLS/SSL, an unencrypted connection is established. The TLS/SSL protocol used is determined by the setting of the Crypto Protocol Version connection option.

Connection Options: Security	Description
	If set to **7 - LoginSSL**, the login request is encrypted using TLS/SSL regardless of whether the server is configured for TLS/SSL. The data is encrypted using TLS/SSL if the server is configured for TLS/SSL, and the data is unencrypted if the server is not configured for TLS/SSL. The TLS/SSL protocol used is determined by the setting of the Crypto Protocol Version connection option. If set to **8 - 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=1) for the connection, regardless of the setting of the Validate Server Certificate option. • You must specify a truststore containing the server certificate against which the server will be validated at connection. **Note:** When establishing a connection to Microsoft Azure Synapse Analytics or Microsoft Analytics Platform System, the driver will enable TLS/SSL data encryption by default (**1 - SSL**). **Default: 0 - None**
Crypto Protocol Version	Specifies the cryptographic protocols to use when TLS/SSL is enabled using the Encryption Method connection option (`EncryptionMethod=1 | 6 | 7 | 8`). **Default: TLSv1.3**,**TLSv1.2** **Note:** TLSv1.3 is only selectable when Encryption Method is set to **8 - Strict**.
Validate Server Certificate	Determines whether the driver validates the certificate that is sent by the database server when TLS/SSL encryption is enabled (`EncryptionMethod=1 | 6 | 7 | 8`). If enabled, 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 Host Name In Certificate option is specified, the driver also validates the certificate using a host name. The Host Name In Certificate option 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 disabled, the driver does not validate the certificate that is sent by the database server. The driver ignores any truststore information specified by the Trust Store and Trust Store Password options. **Default:** Enabled

Connection Options: Security	Description
Enable FIPS	Determines whether the OpenSSL library uses cryptographic algorithms from the FIPS provider or the default provider when TLS/SSL encryption is enabled (`EncryptionMethod=1 | 6 | 7 | 8`). If disabled, the OpenSSL library uses cryptographic algorithms from the default provider. If enabled, the OpenSSL library uses cryptographic algorithms from the FIPS provider. **Default:** Disabled
Trust Store	Specifies either the path and file name of the truststore file or the contents of the TLS/SSL certificates to be used when TLS/SSL is enabled (`EncryptionMethod=1 | 6 | 7 | 8`) and server authentication is used. **Default:** None
Trust Store Password	Specifies the password that is used to access the truststore file when TLS/SSL is enabled (`EncryptionMethod=1 | 6 | 7 | 8`) and server authentication is used. **Default:** None
Host Name In Certificate	A host name for certificate validation when TLS/SSL encryption is enabled (`EncryptionMethod=1 | 6 | 7 | 8`) and validation is enabled (`Validate Server Certificate=1`). **Default:** None
Column Encryption	Specifies whether the driver is enabled for Always Encrypted functionality when accessing data from encrypted columns. 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`, 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 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. **Default: Disabled**

Connection Options: Security	Description
Key Cache Time To Live	Determines whether the driver caches column encryption keys. This option is used when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`). If set to `-1`, the driver caches column encryption keys on a per connection basis. The keys remain in the cache until the connection is closed or the application exits. If set to `0`, the driver does not cache column encryption keys. **Note:** While caching can improve performance, column encryption keys are designed to be deleted periodically from the cache as a security measure. Therefore, we do not recommend caching keys for applications that remain connected for long periods of time. **Default:** `-1`
Key Store Principal Id	Specifies the principal ID used to authenticate against the Azure Key Vault. This option 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, the principal ID and Client Secret must be used to authenticate against the Azure Key Vault. **Note:** The driver currently supports only Azure App Registration as the principal ID. **Default:** None
Key Store Secret	Specifies the Client Secret used to authenticate against the Azure Key Vault. This option is used only when the Always Encrypted feature is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`) and the 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, the Client Secret and principal ID must be used to authenticate against the Azure Key Vault. **Default:** None
AE Keystore Location	Specifies the absolute path to the PKCS #12 file. This option is used only when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`) and the PKCS #12 file is the keystore provider. **Default:** None
AE Keystore Secret	Specifies the password used to access the PKCS #12 file. This option is used only when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`) and a password-protected PKCS #12 file is the keystore provider.

Connection Options: Security	Description
	Default: None

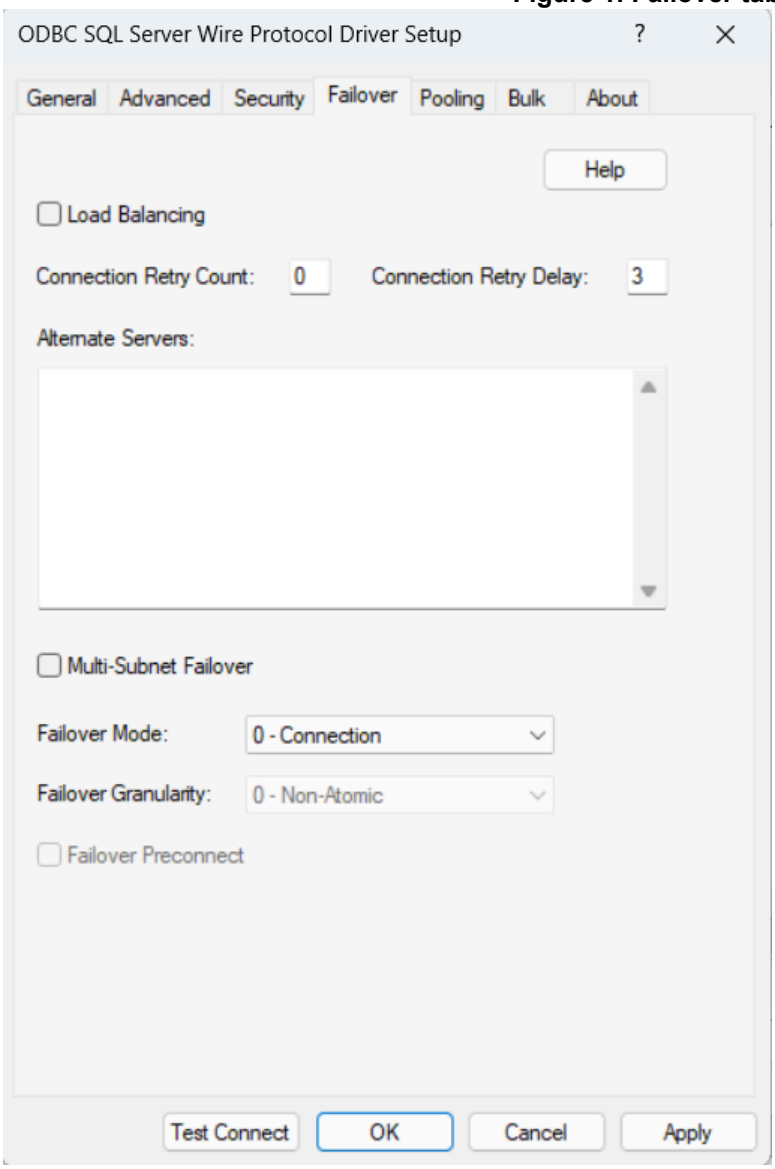
If you finished configuring your driver, proceed to Step 6 in "Data source configuration through a GUI." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- General tab allows you to configure options that are required for creating a data source.
- Advanced tab allows you to configure advanced behavior.
- Failover tab allows you to specify failover data source settings.
- Pooling tab allows you to specify connection pooling settings.
- Bulk tab allows you to specify data source settings for DataDirect Bulk Load.

Failover tab

The Failover tab allows you to specify your failover data source settings. On this tab, provide values for the options in the following table; then, click **Apply**. The fields are optional unless otherwise noted. See "Using failover" for a general description of failover and its related connection options.

Figure 1. Failover tab



Connection Options: Failover	Description
Load Balancing	Determines whether the driver uses client load balancing in its attempts to connect to the database servers (primary and alternate). If enabled, the driver uses client load balancing and attempts to connect to the database servers (primary and alternate servers) in random order. If disabled, 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). **Default:** Disabled

Connection Options: Failover	Description
Connection Retry Count	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. **Default:** `0`
Connection Retry Delay	Specifies the number of seconds the driver waits between connection retry attempts when Connection Retry Count is set to a positive integer. If set to `0`, there is no delay between retries. If set to *x*, the driver waits the specified number of seconds between connection retry attempts. **Default:** `3`
Alternate Servers	A list of alternate database servers to which the driver tries to connect if the primary database server is unavailable. Specifying a value for this option enables connection failover for the driver. The value you specify must be in the form of a string that defines the physical location of each alternate server. All of the other required connection information for each alternate server is the same as what is defined for the primary server connection. For additional information, see "Alternate Servers". **Default:** None
Multi-Subnet Failover	Determines whether the driver attempts parallel connections to the failover IP addresses of an Availability Group during a multi-subnet failover. If set to `1` (Enabled), the driver attempts parallel connections to all failover IP addresses in an Availability Group when the connection is broken or the listener IP address is 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 `0` (Disabled), the driver uses the failover method specified by the Failover Mode connection option when the primary server is unavailable. Use this setting if your environment is not configured for Always On Availability Groups.
Failover Mode	Specifies the type of failover method the driver uses. If set to **0 - Connection**, the driver provides failover protection for new connections only. If set to **1 - Extended Connection**, the driver provides failover protection for new and lost connections, but not any work in progress.

Connection Options: Failover	Description
	If set to **2 - 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.
Failover Granularity	Determines whether the driver fails the entire failover process or continues with the process if errors occur while trying to reestablish a lost connection. If set to **0 - Non-Atomic**, the driver continues with the failover process and posts any errors on the statement on which they occur. If set to **1 - Atomic** the driver fails the entire failover process if an error is generated as the result of anything other than executing and repositioning a Select statement. If an error is generated as a result of repositioning a result set to the last row position, the driver continues with the failover process, but generates a warning that the Select statement must be reissued. If set to **2 - Atomic** Including Repositioning, the driver fails the entire failover process if any error is generated as the result of restoring the state of the connection or the state of work in progress. If set to **3 - Disable Integrity Check**, the driver does not verify that the rows that were restored during the failover process match the original rows. This value applies only when Failover Mode is set to **2 - Select**. **Default: 0 - Non-Atomic**
Failover Preconnect	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. If disabled, the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection. If enabled, 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. **Default:** Disabled

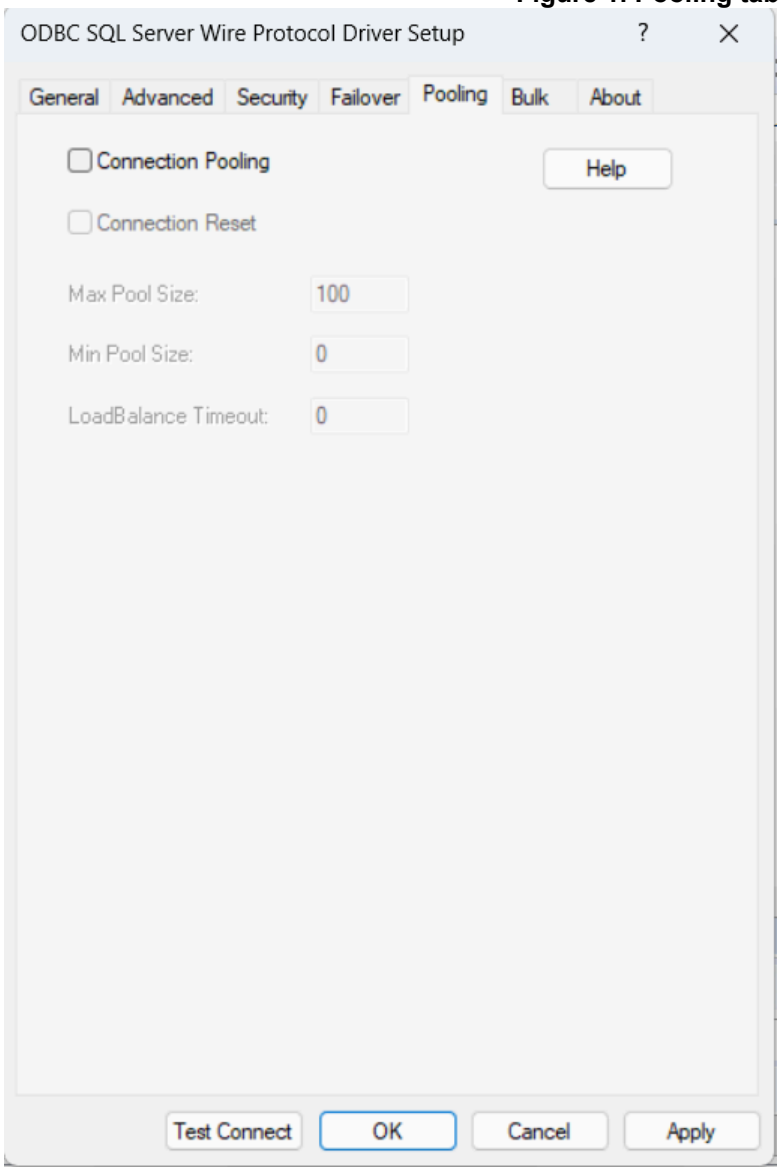
If you finished configuring your driver, proceed to Step 6 in "Data source configuration through a GUI." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- General tab allows you to configure options that are required for creating a data source.
- Advanced tab allows you to configure advanced behavior.
- Security tab allows you to specify security data source settings.
- Pooling tab allows you to specify connection pooling settings.
- Bulk tab allows you to specify data source settings for DataDirect Bulk Load.

Pooling tab

The Pooling Tab allows you to specify your pooling data source settings. On this tab, provide values for the options in the following table; then, click **Apply**. The fields are optional unless otherwise noted. See "Using DataDirect connection pooling" for a general description of connection pooling.

Figure 1. Pooling tab



Connection Options: Pooling	Description
Connection Pooling	Specifies whether to use the driver's connection pooling. If enabled, the driver uses connection pooling. If disabled, the driver does not use connection pooling.

Connection Options: Pooling	Description
	Default: Disabled
Connection Reset	Determines whether the state of connections that are removed from the connection pool for reuse by the application is reset to the initial configuration of the connection. If enabled, the state of connections removed from the connection pool for reuse by an application is reset to the initial configuration of the connection. Resetting the state can negatively impact performance because additional commands must be sent over the network to the server to reset the state of the connection. If disabled, the state of connections is not reset. **Default:** Disabled
Max Pool Size	The maximum number of connections allowed within a single connection pool. When the maximum number of connections is reached, no additional connections can be created in the connection pool. **Default:** `100`
Min Pool Size	The minimum number of connections that are opened and placed in a connection pool, in addition to the active connection, when the pool is created. The connection pool retains this number of connections, even when some connections exceed their Load Balance Timeout value. If set to `0`, no connections are opened in addition to the current existing connection. If set to x, the start-up number of connections in the pool is x in addition to the current existing connection. **Default:** `0`
Load Balance Timeout	Specifies the number of seconds to keep inactive connections open in a connection pool. An inactive connection is a database session that is not associated with an ODBC connection handle, that is, a connection in the pool that is not in use by an application. If set to `0`, inactive connections are kept open. If set to x, inactive connections are closed after the specified number of seconds passes. **Default:** `0` (inactive connections are kept open.)

If you finished configuring your driver, proceed to Step 6 in "Data source configuration through a GUI

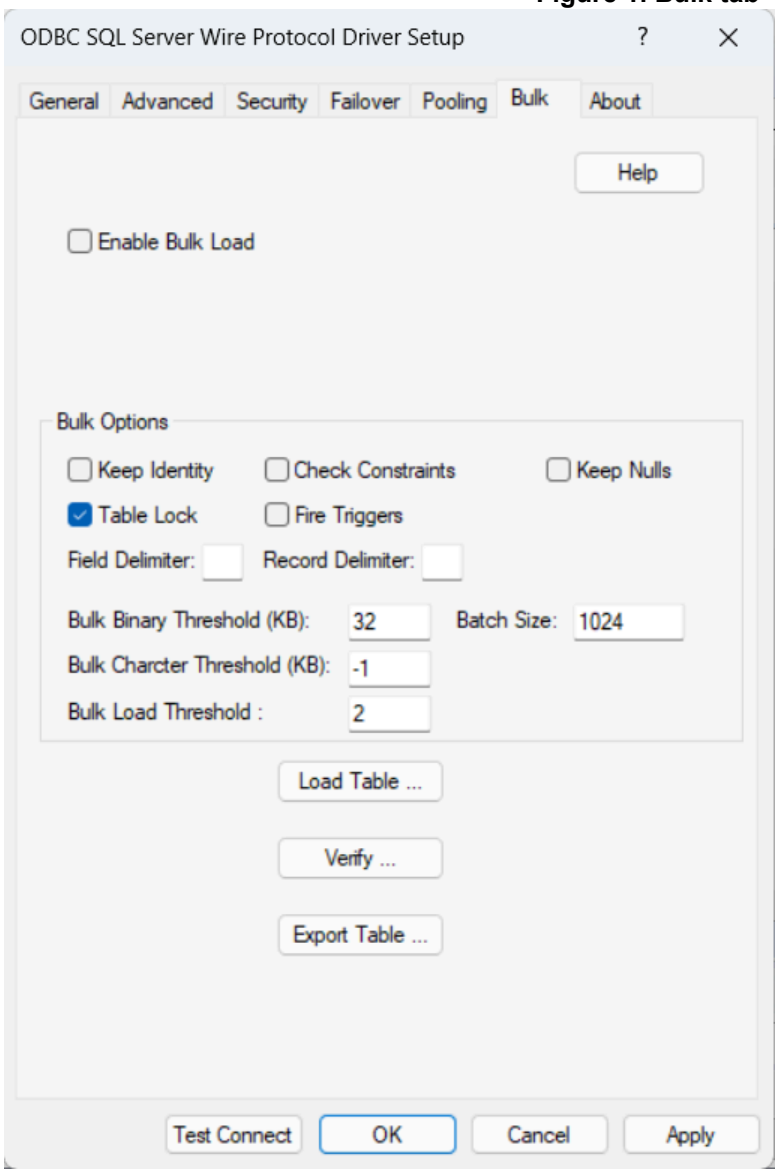
(Windows)." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- General tab allows you to configure options that are required for creating a data source.
- Advanced tab allows you to configure advanced behavior.
- Security tab allows you to specify security data source settings.
- Failover tab allows you to specify failover data source settings.
- Bulk tab allows you to specify data source settings for DataDirect Bulk Load.

Bulk tab

The Bulk tab allows you to specify DataDirect Bulk Load data source settings. On this tab, provide values for the options in the following table; then, click **Apply**. The fields are optional unless otherwise noted. See "Using DataDirect Bulk Load" for more information.

Figure 1. Bulk tab



Connection Options: Bulk	Description
Enable Bulk Load	Specifies the bulk load method. If enabled, 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 disabled, the driver uses standard parameter arrays. **Default:** Disabled

Connection Options: Bulk	Description
Bulk Options	Toggles options for the bulk load process. **Keep Identity** - Preserves source identity values. When not enabled, identity values are assigned by the destination. **Check Constraints** - Checks constraints while data is being inserted. **Keep Nulls** - Preserves null values in the destination table regardless of the settings for default values. When not enabled, null values are replaced by column default values, where applicable. **Table Lock** - Assigns a table lock for the duration of the bulk copy operation. Other applications are not permitted to update the table during the copy operation. When not enabled, the default bulk locking mechanism (row or table) specified by the table lock on bulk load server option is used. **Fire Triggers** - Causes the server to fire the insert triggers for rows being inserted into the database. **Default:** Table Lock enabled
Field Delimiter	Specifies the character that the driver will use to delimit the field entries in a bulk load data file. **Default:** None
Record Delimiter	Specifies the character that the driver will use to delimit the record entries in a bulk load data file. **Default:** None
Bulk Binary Threshold	The maximum size, in KB, of binary data that is exported to the bulk data file. If set to `-1`, all binary data, regardless of size, is written to the bulk data file, not to an external file. If set to `0`, all binary data, regardless of size, is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file. If set to *x*, any binary data exceeding this specified number of KB is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file. **Default:** None
Batch Size	The number of rows that the driver sends to the database at a time during bulk operations. This value applies to all methods of bulk loading.

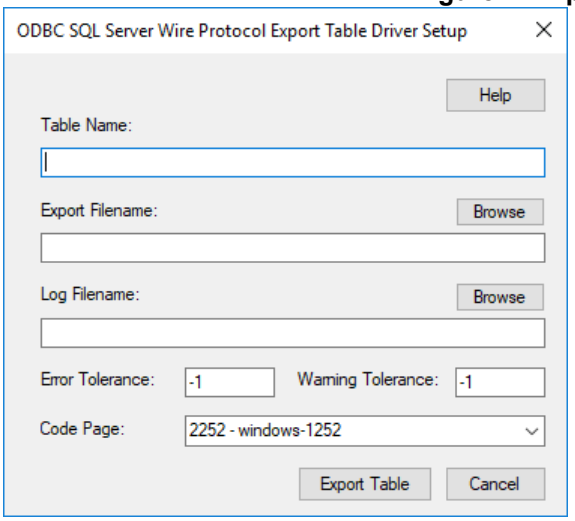
Connection Options: Bulk	Description
	Default: `1024`
Bulk Character Threshold	The maximum size, in KB, of character data that is exported to the bulk data file. If set to `-1`, all character data, regardless of size, is written to the bulk data file, not to an external file. If set to `0`, all character data regardless of size, is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file. If set to *x*, any character data exceeding this specified number of KB is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file. Default: `-1`
Bulk Load Threshold	Determines when the driver uses bulk load for insert, update, delete, or batch operations. If the Enable Bulk Load option is enabled and the number of rows affected by an insert, update, delete, or batch operation exceeds the threshold specified by this option, the driver uses SQL Server bulk load protocol to perform the operation. If set to `0`, the driver always uses bulk load to execute insert, update, delete, or batch operations. If set to *x*, the driver only uses bulk load if the Enable Bulk Load option is enabled and the number of rows to be updated by an insert, update, delete, or batch operation exceeds the threshold. If the operation times out, the driver returns an error. **Default:** `2`

If your application is already coded to use parameter array batch functionality, you can leverage the database bulk load protocol through the Enable Bulk Load connection option. Enabling this option automatically converts the parameter array batch operation to use the database bulk load protocol.

If you are not using parameter array batch functionality, you can export data to a bulk load data file, verify the metadata of the bulk load configuration file against the structure of the target table, and bulk load data to a table. Use the following steps to accomplish these tasks.

1. To export data from a table to a bulk load data file, click **Export Table** from the Bulk tab. The **Export Table** dialog box appears.

Figure 2. Export Table dialog box



Both a bulk data file and a bulk configuration file are produced by exporting a table. The configuration file has the same name as the data file, but with an XML extension. See "Using DataDirect Bulk Load" for details about these files.

The bulk export operation can create a log file and can also export to external files. See "External overflow files" for more information. The export operation can be configured such that if any errors or warnings occur:

- The operation always completes.
- The operation always terminates.
- The operation terminates after a certain threshold of warnings or errors is exceeded.

Table Name: A string that specifies the name of the source database table containing the data to be exported.

Export Filename: A string that specifies the path (relative or absolute) and file of the bulk load data file to which the data is to be exported. It also specifies the file name of the bulk configuration file. The file name must be the fully qualified path to the bulk data file. These files must not already exist; if one of both of them already exists, an error is returned.

Log Filename: A string that specifies the path (relative or absolute) and file name of the bulk log file. The log file is created if it does not exist. The file name must be the fully qualified path to the log file. Events logged to this file are:

- Total number of rows fetched
- A message for each row that failed to export
- Total number of rows that failed to export
- Total number of rows successfully exported

Information about the load is written to this file, preceded by a header. Information about the next load is appended to the end of the file.

If you do not supply a value for Log Filename, no log file is created.

Error Tolerance: A value that specifies the number of errors to tolerate before an operation terminates. A value of `0` indicates that no errors are tolerated; the operation fails when the first error is encountered.

The default of `-1` means that an infinite number of errors is tolerated.

Warning Tolerance: A value that specifies the number of warnings to tolerate before an operation terminates. A value of `0` indicates that no warnings are tolerated; the operation fails when the first warning is encountered.

The default of `-1` means that an infinite number of warnings is tolerated.

Code Page: A value that specifies the code page value to which the driver must convert all data for storage in the bulk data file. See "Character set conversions" for more information.

The default value on Windows is the current code page of the machine. On UNIX/Linux/macOS, the default value is `4` (ISO 8559-1 Latin-1).

Click **Export Table** to connect to the database and export data to the bulk data file or click **Cancel**.

Click **Export Table** to connect to the database and export data to the bulk data file or click **Cancel**.

2. To verify the metadata of the bulk load configuration file against the structure of the target database table, click **Verify** from the Bulk tab. See "Verification of the bulk load configuration file" for details. The **ODBC SQL Server Wire Protocol Verify Driver Setup** dialog box appears.

Figure 3. ODBC SQL Server Wire Protocol Verify Driver Setup dialog box

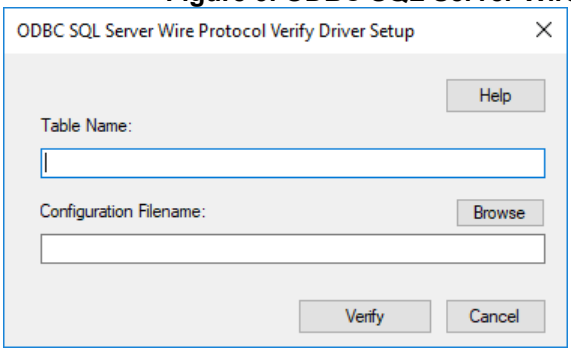


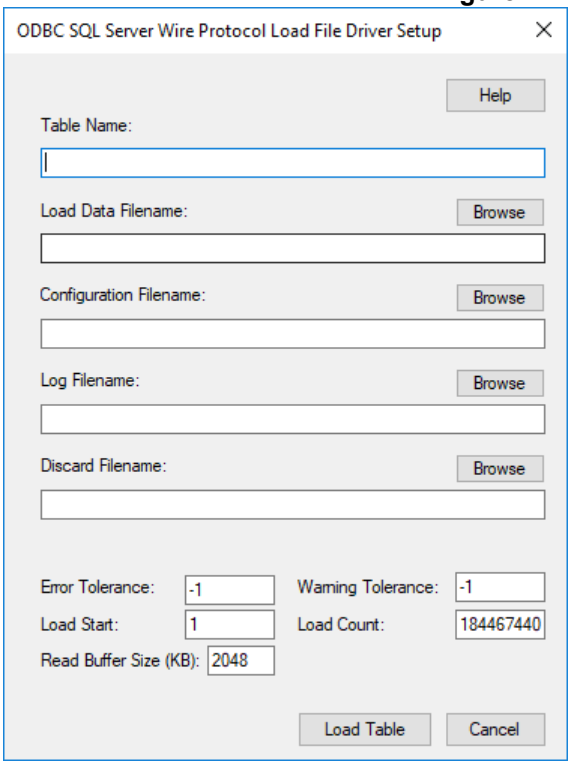
Table Name: A string that specifies the name of the target database table into which the data is to be loaded.

Configuration Filename: A string that specifies the path (relative or absolute) and file name of the bulk configuration file. The file name must be the fully qualified path to the configuration file.

Click **Verify** to verify table structure or click **Cancel**.

3. To bulk load data from the bulk data file to a database table, click **Load Table** from the Bulk tab. The **Load File** dialog box appears.

Figure 4. Load File dialog box



The load operation can create a log file and can also create a discard file that contains rows rejected during the load. The discard file is in the same format as the bulk load data file. After fixing reported issues in the discard file, the bulk load can be reissued using the discard file as the bulk load data file.

The export operation can be configured such that if any errors or warnings occur:

- The operation always completes.
- The operation always terminates.
- The operation terminates after a certain threshold of warnings or errors is exceeded.

If a load fails, the Load Start and Load Count options can be used to control which rows are loaded when a load is restarted after a failure.

Table Name: A string that specifies the name of the target database table into which the data is loaded.

Load Data Filename: A string that specifies the path (relative or absolute) and file name of the bulk data file from which the data is loaded. The file name must be the fully qualified path to the bulk data file.

Configuration Filename: A string that specifies the path (relative or absolute) and file name of the bulk configuration file. The file name must be the fully qualified path to the configuration file.

Log Filename: A string that specifies the path (relative or absolute) and file name of the bulk log file. The file name must be the fully qualified path to the log file. Specifying a value for Log Filename creates the file if it does not already exist. Events logged to this file are:

- Total number of rows read
- Message for each row that failed to load

- Total number of rows that failed to load
- Total number of rows successfully loaded

Information about the load is written to this file, preceded by a header. Information about the next load is appended to the end of the file.

If you do not specify a value for Log Filename, no log file is created.

Discard Filename: A string that specifies the path (relative or absolute) and file name of the bulk discard file. The file name must be the fully qualified path to the discard file. Any row that cannot be inserted into database as result of bulk load is added to this file, with the last row rejected added to the end of the file.

Information about the load is written to this file, preceded by a header. Information about the next load is appended to the end of the file.

If you do not specify a value for Discard Filename, a discard file is not created.

Error Tolerance: A value that specifies the number of errors to tolerate before an operation terminates. A value of `0` indicates that no errors are tolerated; the operation fails when the first error is encountered.

The default of `-1` means that an infinite number of errors is tolerated.

Load Start: A value that specifies the first row to be loaded from the data file. Rows are numbered starting with 1. For example, when Load Start is 10, the first 9 rows of the file are skipped and the first row loaded is row 10. This option can be used to restart a load after a failure.

The default value is `1`.

Read Buffer Size (KB): A value that specifies the size, in KB, of the buffer that is used to read the bulk data file for a bulk load operation.

The default value is `2048`.

Warning Tolerance: A value that specifies the number of warnings to tolerate before an operation terminates. A value of `0` indicates that no warnings are tolerated; the operation fails when the first warning is encountered.

The default of `-1` means that an infinite number of warnings is tolerated.

Load Count: A value that specifies the number of rows to be loaded from the data file. The bulk load operation loads rows up to the value of Load Count from the file to the database. It is valid for Load Count to specify more rows than exist in the data file. The bulk load operation completes successfully when either the number of rows specified by the Load Count value has been loaded or the end of the data file is reached. This option can be used in conjunction with Load Start to restart a load after a failure.

The default value is the maximum value for SQLULEN. If set to `0`, no rows are loaded.

Click **Load Table** to connect to the database and load the table or click **Cancel**.

If you finished configuring your driver, proceed to Step 6 in "Data source configuration through a GUI (Windows)." Optionally,you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- General tab allows you to configure options that are required for creating a data source.
- Advanced tab allows you to configure advanced behavior.
- Security tab allows you to specify security data source settings.
- Failover tab allows you to specify failover data source settings.
- Pooling tab allows you to specify connection pooling settings.

Using a connection string

If you want to use a connection string for connecting to a database, or if your application requires it, you must specify either a DSN (data source name), a File DSN, or a DSN-less connection in the string. The difference is whether you use the `DSN=`, `FILEDSN=`, or the `DRIVER=` keyword in the connection string, as described in the ODBC specification. A DSN or FILEDSN connection string tells the driver where to find the default connection information. Optionally, you may specify *attribute=value* pairs in the connection string to override the default values stored in the data source.

The DSN connection string has the form:

```
DSN=data_source_name[;attribute=value[;attribute=value]...]
```

The FILEDSN connection string has the form:

```
FILEDSN=filename.
dsn
[;attribute=value[;attribute=value]...]
```

The DSN-less connection string specifies a driver instead of a data source. All connection information must be entered in the connection string because the information is not stored in a data source.

The DSN-less connection string has the form:

```
DRIVER=[{]driver_name[}][;attribute=value[;attribute=value]...]
```

"Connection option descriptions" lists the long and short names for each attribute, as well as the initial default value when the driver is first installed. You can specify either long or short names in the connection string.

An example of a DSN connection string with overriding attribute values for SQL Server for Linux/UNIX/ Windows is:

```
DSN=ACCOUNTING;DATABASE=ACCT
```

A FILEDSN connection string is similar except for the initial keyword:

```
FILEDSN=SQLServer.dsn;DATABASE=ACCT
```

A DSN-less connection string must provide all necessary connection information:

```
DRIVER={DataDirect 8.0 SQL Server Wire Protocol};HOST=SQLServer1;PORT=1433;
UID=JOHN;PWD=XYZZY;DB=SQLSdb1
```

Password Encryption Tool (UNIX/Linux only)

On UNIX and Linux, Progress DataDirect provides a Password Encryption Tool, called ddencpwd, that encrypts passwords for secure handling in connection strings and odbc.ini files. At connection, the driver decrypts these passwords and passes them to the data source as required. Passwords can be encrypted for any option, including:

- KeyPassword
- KeyStorePassword
- TrustStorePassword
- Password

To use the Password Encryption Tool:

1. From a command line, navigate to the directory containing the ddencpwd application. By default, this is *install_directory*/tools.
2. Enter the following command:

   ddencpwd *password*

   where:

   password

   is the password you want to encrypt.
3. The tool returns an encrypted password value. Specify the returned value for the corresponding attribute in the connection string or odbc.ini file. For example, if you encrypted the password for KeyPassword, specify the following in your connection string or datasource definition:

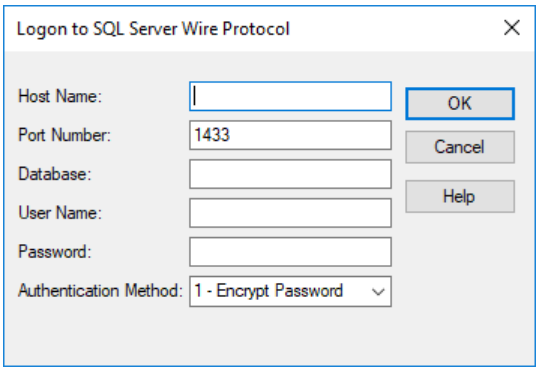
   KeyPassword=*returned_value*
4. Repeat Steps 2 and 3 to encrypt additional passwords.
5. If using an odbc.ini file, save your file.

This completes this tutorial. You are now ready to connect using encrypted passwords.

Using a logon dialog box

Some ODBC applications display a Logon dialog box when you are connecting to a data source. In these cases, the data source name has already been specified.

Note: The Logon Dialog is not displayed if Authentication Mode has previously been set to Kerberos and the Host Name is specified in the data source.



In the Logon dialog box, provide the following information:

1. Type an IP address in Host Name in following format: *IP_address*. For example, you can enter `199.226.224.34`.

   The IP address can be specified in IPv4 on Windows, and in either IPv4 or IPv6 format, or a combination of the two, on UNIX. See "Using IP addresses" for details about these formats.

   If your network supports named servers, you can specify an address as: *server_name*. For example, you can enter `SSserver`.

   To specify a named instance of Microsoft SQL Server, use the format: *server_name\instance_name*. If only a server name is specified with no instance name, the driver uses the default instance on the server.

2. Type the Port Number of the server listener. The default is `1433`.
3. Type the name of the database to which you want to connect. If you do not specify a value, the default database that is defined by Microsoft SQL Server is used.
4. Type your Microsoft SQL Server login ID.
5. Type your password.
6. Select an Authentication Method:

   If set to **1 - Encrypt Password**, the driver sends the user ID in clear text and an encrypted password to the server for authentication.

   If set to **4 - Kerberos**, the driver uses Kerberos authentication. This method supports both Windows Active Directory Kerberos and MIT Kerberos environments.

   If set to **13 - Active Directory Password**, the driver authenticates using an Entra ID user name and password when connecting to a Azure SQL Database data store. All communications between the service are encrypted using TLS/SSL.

   If set to **36 - Active Directory Service Principal**, the driver authenticates using an Entra ID service principal when establishing a connection to an Azure SQL Database data store. All communications between the service are encrypted using TLS/SSL.

   If set to **37 - Active Directory Manged Identity**, the driver authenticates using a managed identity when accessing Azure resources. All communications between the service are encrypted using TLS/SSL.

7. Click **OK** to complete the logon and to update the values in the Registry.

Performance considerations

The following connection options can affect driver performance.

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

- **Column Encryption (ColumnEncryption)**: Due to the overhead associated with encrypting and decrypting data, the Always Encrypted functionality can adversely affect performance when enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`). You can mitigate the performance impact of Always Encrypted depending on your use case:
  - If your application only needs to retrieve and decrypt columns, you can improve performance over the behavior of the `Enabled` setting by specifying a value of `ResultsetOnly`. In this setting, Always Encrypted behavior is enabled only when returning result sets. Queries containing parameters that affect encrypted columns will return an error.
  - If your application only occasionally needs to encrypt or decrypt columns, you can override the Column Encryption option by setting a value for the `SQL_SOPT_SS_COLUMN_ENCRYPTION` statement attribute for a statement. `SQL_SOPT_SS_COLUMN_ENCRYPTION` allows you to toggle support for Always Encrypted without having to establish a new connection, thereby reducing the performance impact when you are not using the feature. Note that the statement attribute is only supported when `ColumnEncryption=Enabled` | `ResultsetOnly`. See "Always Encrypted" for details.
- **Key Cache Time To Live (AEKeyCacheTTL)**: When Always Encrypted functionality is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`), you can determine whether column encryption keys are cached using the Key Cache Time To Live option. Caching column encryption keys can provide performance gains by eliminating the overhead associated with fetching and decrypting keys for the same data multiple times during a connection. Note that column encryption keys are designed to be deleted from the cache as a security measure. Therefore, we do not recommend caching column encryption keys for applications that remain connected for long periods of time.

Connection Pooling (Pooling): If you enable the driver to use connection pooling, you can set additional options that affect performance:

- **Load Balance Timeout (LoadBalanceTimeout)**: You can define how long to keep connections in the pool. The time that a connection was last used is compared to the current time and, if the timespan exceeds the value of the Load Balance Timeout option, the connection is destroyed. The Min Pool Size option can cause some connections to ignore this value.
- **Connection Reset (ConnectionReset)**: Resetting a re-used connection to the initial configuration settings impacts performance negatively because the connection must issue additional commands to the server.
- **Max Pool Size (MaxPoolSize)**: Setting the maximum number of connections that the pool can contain too low might cause delays while waiting for a connection to become available. Setting the number too high wastes resources.
- **Min Pool Size (MinPoolSize)**: A connection pool is created when the first connection with a unique connection string connects to the database. The pool is populated with connections up to the minimum pool size, if one has been specified. The connection pool retains this number of connections, even when some connections exceed their Load Balance Timeout value.

Enable Bulk Load (**EnableBulkLoad**): If your application performs bulk loading of data, you can improve performance by configuring the driver to use the database system's bulk load functionality instead of database array binding. The trade-off to consider for improved performance is that using the bulk load functionality can bypass data integrity constraints.

Enable Server Side Cursors (**EnableServersideCursors**): Employing scrollable cursors are more expensive than using forward-only cursors, and, therefore, can adversely impact performance. If your application does not always require the use of scrollable cursors, you can restrict the use of server-side scrollable cursors using the Enable Server Side Cursors option. For best performance, you can disable all server-side scrollable cursors by setting `EnableServersideCursors=0`.

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

Failover Mode (FailoverMode): Although high availability that replays queries after a failure provides increased levels of protection, it can adversely affect performance because of increased overhead.

Packet Size (**PacketSize**): Typically, it is optimal for the client to use the maximum packet size that the database 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 the PacketSize attribute is set to the maximum packet size of the server.

Use Snapshot Transactions (**SnapshotSerializable**): You must have your Microsoft SQL Server 2005 and higher database configured for snapshot isolation for this connection option to work. 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 2005 and higher 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 option is enabled, performance is improved due to increased concurrency.

See "Using the Snapshot isolation level" for details.

Using failover

To ensure continuous, uninterrupted access to data, the Progress DataDirect *for* ODBC driver provides the following levels of failover protection, listed from basic to more comprehensive:

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

The method you choose depends on how failure tolerant your application is. For example, if a communication failure occurs while processing, can your application handle the recovery of transactions and restart them? Your application needs the ability to recover and restart transactions when using either extended connection failover mode or select connection failover mode. The advantage of select mode is that

it preserves the state of any work that was being performed by the Select statement at the time of connection loss. If your application had been iterating through results at the time of the failure, when the connection is reestablished the driver can reposition on the same row where it stopped so that the application does not have to undo all of its previous result processing. For example, if your application were paging through a list of items on a Web page when a failover occurred, the next page operation would be seamless instead of starting from the beginning. Performance, however, is a factor in selecting a failover mode. Select mode incurs additional overhead when tracking what rows the application has already processed.

You can specify which failover method you want to use by setting the Failover Mode (FailoverMode) connection option. Read the following sections for details on each failover method:

- Connection Failover
- Extended Connection Failover
- Select Connection Failover

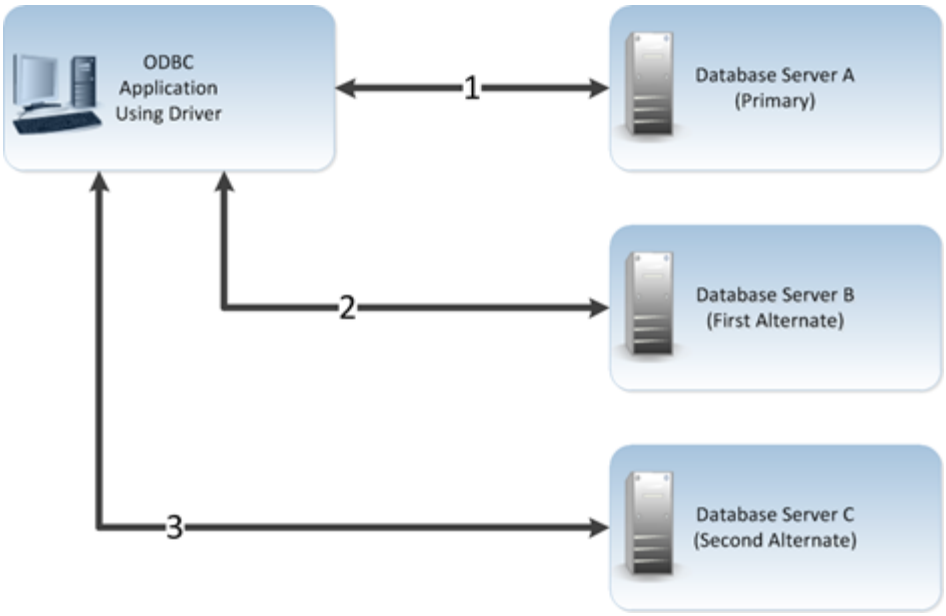
Regardless of the failover method you choose, you must configure one or multiple alternate servers using the Alternate Servers connection option. See "Guidelines for primary and alternate servers" for information about primary and alternate servers.

Connection failover

Connection failover allows an application to connect to an alternate, or backup, database server if the primary database server is unavailable, for example, because of a hardware failure or traffic overload. Connection failover provides failover protection for new connections only and does not provide protection for lost connections to the database, nor does it preserve states for transactions or queries.

You can customize the driver for connection failover by configuring a list of alternate database servers that are tried if the primary server is not accepting connections. Connection attempts continue until a connection is successfully established or until all the alternate database servers have been tried the specified number of times.

For example, suppose you have the environment shown in the following illustration with multiple database servers: Database Server A, B, and C. Database Server A is designated as the primary database server, Database Server B is the first alternate server, and Database Server C is the second alternate server.



First, the application attempts to connect to the primary database server, Database Server A (**1**). If connection failover is enabled and Database Server A fails to accept the connection, the application attempts to connect to Database Server B (**2**). If that connection attempt also fails, the application attempts to connect to Database Server C (**3**).

In this scenario, it is probable that at least one connection attempt would succeed, but if no connection attempt succeeds, the driver can retry each alternate database server (primary and alternate) for a specified number of attempts. You can specify the number of attempts that are made through the *connection retry* feature. You can also specify the number of seconds of delay, if any, between attempts through the *connection delay* feature. See "Using connection retry" for more information.

A driver fails over to the next alternate database server only if a successful connection cannot be established with the current alternate server. If the driver successfully establishes communication with a database server and the connection request is rejected by the database server because, for example, the login information is invalid, then the driver generates an error and does not try to connect to the next database server in the list. It is assumed that each alternate server is a mirror of the primary and that all authentication parameters and other related information are the same.

For details on configuring connection failover for your driver, see "Configuring failover-related options."

Extended connection failover

Extended connection failover provides failover protection for the following types of connections:

- New connections, in the same way as described in "Connection failover"
- Lost connections

When a connection to the database is lost, the driver fails over the connection to an alternate server, restoring the same state of the connection at the time it was lost. For example, when reestablishing a lost connection on the alternate database server, the driver performs the following actions:

- Restores the connection using the same connection options specified by the lost connection

- Reallocates statement handles and attributes
- Logs in the user to the database with the same user credentials
- Restores any prepared statements associated with the connection and repopulates the statement pool
- Restores manual commit mode if the connection was in manual commit mode at the time of the failover

The driver does not preserve work in progress. For example, if the database server experienced a hardware failure while processing a query, partial rows processed by the database and returned to the client would be lost. If the driver was in manual commit mode and one or more Inserts or Updates were performed in the current transaction before the failover occurred, then the transaction on the primary server is rolled back. The Inserts or Updates done before the failover are not committed to the primary server. Your application needs to rerun the transaction after the failover because the Inserts or Updates done before the failover are not repeated by the driver on the failover connection.

When a failover occurs, if a statement is in allocated or prepared state, the next operation on the statement returns a SQL state of 01000 and a vendor code of 0. If a statement is in an executed or prepared state, the next operation returns a SQL state of 40001 and a vendor code of 0. Either condition returns an error message similar to:

```
Your connection has been terminated. However, you have been successfully
connected to
the next available AlternateServer: 'HOSTNAME=Server4:PORTNUMBER=
1521:SERVICENAME=test'.
All active transactions have been rolled back.
```

The driver retains all connection settings made through ODBC API calls when a failover connection is made. It does not, however, retain any session settings established through SQL statements. This can be done through the Initialization String connection option, described in the individual driver chapters.

The driver retains the contents of parameter buffers, which can be important when failing over after a fetch. All Select statements are re-prepared at the time the failover connection is made. All other statements are placed in an allocated state.

If an error occurs while the driver is reestablishing a lost connection, the driver can fail the entire failover process or proceed with the process as far as it can. For example, suppose an error occurred while reestablishing the connection because a table for which the driver had a prepared statement did not exist on the alternate connection. In this case, you may want the driver to notify your application of the error and proceed with the failover process. You can choose how you want the driver to behave if errors occur during failover by setting the Failover Granularity connection option.

During the failover process, your application may experience a short pause while the driver establishes a connection on an alternate server. If your application is time-sensitive (a real-time customer order application, for example) and cannot absorb this wait, you can set the "Failover preconnect" connection option to true. Setting the Failover Preconnect option to true instructs the driver to establish connections to the primary server and an alternate server at the same time. Your application uses the first connection that is successfully established. If this connection to the database is lost at a later time, the driver saves time in reestablishing the connection on the server to which it fails over because it can use the spare connection in its failover process.

This pre-established failover connection is not used by the driver until the driver determines that it needs to fail over. If the server to which the driver is connected or the network equipment through which the connection is routed is configured with a timeout, the pre-configured failover connection could time out. The pre-configured failover connection can also be lost if the failover server is brought down and back up again. The driver tries to establish the connection to the failover server again if the connection is lost.

Select connection failover

Select connection failover provides failover protection for the following types of connections:

- New connections, in the same way as described in "Connection failover"
- Lost connections, in the same way as described in "Extended connection failover"

In addition, the driver can recover work in progress because it keeps track of the last Select statement the application executed on each Statement handle, including how many rows were fetched to the client. For example, if the database had only processed 500 of 1,000 rows requested by a Select statement when the connection was lost, the driver would reestablish the connection to an alternate server, re-execute the Select statement, and position the cursor on the next row so that the driver can continue fetching the balance of rows as if nothing had happened.

Performance, however, is a factor when considering whether to use Select mode. Select mode incurs additional overhead when tracking what rows the application has already processed.

The driver only recovers work requested by Select statements. You must explicitly restart the following types of statements after a failover occurs:

- Insert, Update, or Delete statements
- Statements that modify the connection state, for example, SET or ALTER SESSION statements
- Objects stored in a temporary tablespace or global temporary table
- Partially executed stored procedures and batch statements

When in manual transaction mode, no statements are rerun if any of the operations in the transaction were Insert, Update, or Delete. This is true even if the statement in process at the time of failover was a Select statement.

By default, the driver verifies that the rows that are restored match the rows that were originally fetched and, if they do not match, generates an error warning your application that the Select statement must be reissued. By setting the Failover Granularity connection option, you can customize the driver to ignore this check altogether or fail the entire failover process if the rows do not match.

When the row comparison does not agree, the default behavior of Failover Granularity returns a SQL state of 40003 and an error message similar to:

```
Unable to position to the correct row after a successful failover attempt to
AlternateServer: 'HOSTNAME=Server4:PORTNUMBER= 1521:SERVICENAME=test'. You must
reissue the select statement.
```

If you have configured Failover Granularity to fail the entire failover process, the driver returns a SQL state of 08S01 and an error message similar to:

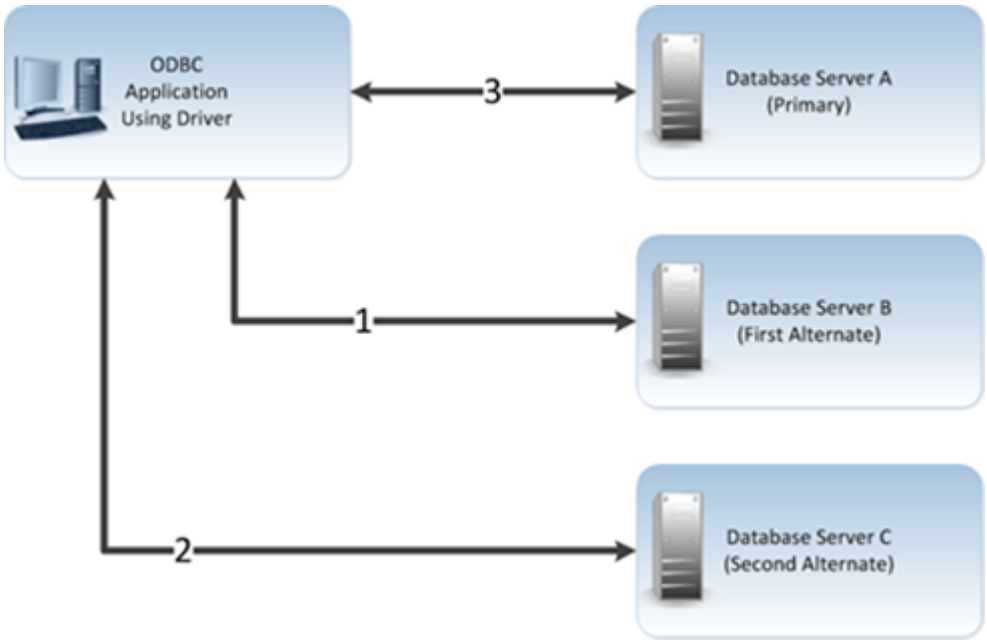
```
Your connection has been terminated and attempts to complete the failover
process to the following Alternate Servers have failed: AlternateServer:
'HOSTNAME=Server4:PORTNUMBER= 1521:SERVICENAME=test'. All active transactions
have been rolled back.
```

Guidelines for primary and alternate servers

SQL Server databases provide advanced database replication technologies through the AlwaysOn feature. The failover functionality provided by the drivers does not require AlwaysOn, but can work with this technology to provide comprehensive failover protection. To ensure that failover works correctly, alternate servers should mirror data on the primary server or be part of a configuration where multiple database nodes share the same physical data.

Using client load balancing

Client load balancing helps distribute new connections in your environment so that no one server is overwhelmed with connection requests. When client load balancing is enabled, the order in which primary and alternate database servers are tried is random. For example, suppose that client load balancing is enabled as shown in the following illustration:



First, Database Server B is tried (**1**). Then, Database Server C may be tried (**2**), followed by a connection attempt to Database Server A (**3**). In contrast, if client load balancing were not enabled in this scenario, each database server would be tried in sequential order, primary server first, then each alternate server based on its entry order in the alternate servers list.

Client load balancing is controlled by the Load Balancing connection option. For details on configuring client load balancing, see the appropriate driver chapter in this book.

Using Connection Retry

Connection retry defines the number of times the driver attempts to connect to the primary server and, if configured, alternate database servers after the initial unsuccessful connection attempt. It can be used with connection failover, extended connection failover, and select failover. Connection retry can be an important

strategy for system recovery. For example, suppose you have a power failure in which both the client and the server fails. When the power is restored and all computers are restarted, the client may be ready to attempt a connection before the server has completed its startup routines. If connection retry is enabled, the client application can continue to retry the connection until a connection is successfully accepted by the server.

Connection retry can be used in environments that have only one server or can be used as a complementary feature with connection failover in environments with multiple servers.

Using the connection options Connection Retry Count and Connection Retry Delay, you can specify the number of times the driver attempts to connect and the time in seconds between connection attempts. For details on configuring connection retry, see "Configuring failover-related options."

Configuring failover-related options

The following table summarizes how failover-related connection options work with the driver. See "Connection option descriptions" for details about configuring the options. The step numbers in the table refer the procedure that follows the table

Table 3: Summary: Failover and Related Connection Options

Option	Characteristic
Alternate Servers (AlternateServers) (See step 1)	One or multiple alternate database servers. An IP address or server name identifying each server is required. **Default:** None
Connection Retry Count (ConnectionRetryCount) (See step 5)	Number of times the driver retries the primary database server, and if specified, alternate servers until a successful connection is established. **Default:** `0`
Connection Retry Delay (ConnectionRetryDelay) (See step 6)	Wait interval, in seconds, between connection retry attempts when the Connection Retry Count option is set to a positive integer. **Default:** `3`
Failover Granularity (FailoverGranularity) (See step 3)	Determines whether the driver fails the entire failover process or continues with the process if errors occur while trying to reestablish a lost connection. If set to `0` (Non-Atomic), the driver continues with the failover process and posts any errors on the statement on which they occur. If set to `1` (Atomic) the driver fails the entire failover process if an error is generated as the result of anything other than executing and repositioning a Select statement. If an error is generated as a result of repositioning a result set to the last row position, the driver continues with the failover process, but generates a warning that the Select statement must be reissued. If set to `2` (Atomic Including Repositioning), the driver fails the entire failover process if

Option	Characteristic
	any error is generated as the result of restoring the state of the connection or the state of work in progress. If set to `3` (Disable Integrity Check), the driver does not verify that the rows that were restored during the failover process match the original rows. This value applies only when Failover Mode is set to `2` (Select). **Default:** `0` (Non-Atomic)
Failover Mode (FailoverMode) (See step 2)	Specifies the type of failover method the driver uses. If set to `0` (Connection), the driver provides failover protection for new connections only. If set to `1` (Extended Connection), the driver provides failover protection for new and lost connections, but not any work in progress. If set to `2` (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. **Default:** `0` (Connection)
Failover Preconnect (FailoverPreconnect) (See step 4)	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. If set to `0` (Disabled), 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. If set to `1` (Enabled), 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. **Default:** `0` (Disabled)
Load Balancing (LoadBalancing) (See step 7)	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 Alternate Servers option. If set to `1` (Enabled), the driver uses client load balancing and attempts to connect to the database servers (primary and alternate servers) in random order. If set to `0` (Disabled), 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).

Option	Characteristic
	Default: `0` (Disabled)

1. To configure connection failover, you **must** specify one or more alternate database servers that are tried at connection time if the primary server is not accepting connections. To do this, use the Alternate Servers connection option. Connection attempts continue until a connection is successfully established or until all the database servers in the list have been tried once (the default).
2. Choose a failover method by setting the Failover Mode connection option. The default method is Connection (`FailoverMode=0`).
3. If Failover Mode is Extended Connection (`FailoverMode=1`) or Select (`FailoverMode=2`), set the Failover Granularity connection option to specify how you want the driver to behave if errors occur while trying to reestablish a lost connection. The default behavior of the driver is Non-Atomic (`FailoverGranularity=0`), which continues with the failover process and posts any errors on the statement on which they occur. Other values are:

   Atomic (`FailoverGranularity=1`): the driver fails the entire failover process if an error is generated as the result of anything other than executing and repositioning a Select statement. If an error is generated as a result of repositioning a result set to the last row position, the driver continues with the failover process, but generates a warning that the Select statement must be reissued.

   Atomic including Repositioning (`FailoverGranularity=2`): the driver fails the entire failover process if any error is generated as the result of restoring the state of the connection or the state of work in progress.

   Disable Integrity Check (`FailoverGranularity=3`): the driver does not verify that the rows restored during the failover process match the original rows. This value applies only when Failover Mode is set to Select (`FailoverMode=2`).

4. Optionally, enable the Failover Preconnect connection option (`FailoverPreconnect=1`) if you want the driver to establish a connection with the primary and an alternate server at the same time. This value applies only when Failover Mode is set to Extended Connection (`FailoverMode=1`) or Select (`FailoverMode=2`). 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=0`).
5. Optionally, specify the number of times the driver attempts to connect to the primary and alternate database servers after the initial unsuccessful connection attempt. By default, the driver does not retry. To set this feature, use the Connection Retry Count connection option.
6. Optionally, specify the wait interval, in seconds, between attempts to connect to the primary and alternate database servers. The default interval is 3 seconds. To set this feature, use the Connection Retry Delay connection option.
7. Optionally, specify whether the driver will use client load balancing in its attempts to connect to primary and alternate database servers. If load balancing is enabled, the driver uses a random pattern instead of a sequential pattern in its attempts to connect. The default value is not to use load balancing. To set this feature, use the Load Balancing connection option.

A connection string example

The following connection string configures the driver to use connection failover in conjunction with some of its optional features.

```
DSN=AcctSQLServer;AlternateServers=(HostName=mySQLServer:PortNumber=1433:Databa
se=Accounting,
HostName=255.201.11.24:PortNumber=1434:Database=Accounting);
ConnectionRetryCount=4;ConnectionRetryDelay=5;LoadBalancing=1;FailoverMode=0
```

Specifically, this connection string configures the driver to use two alternate servers as connection failover servers, to attempt to connect four additional times if the initial attempt fails, to wait five seconds between attempts, to try the primary and alternate servers in a random order, and to attempt reconnecting on new connections only. The additional connection information required for the alternate servers is specified in the data source AcctSQLServer.

An odbc.ini file example

To configure the 32-bit driver to use connection failover in conjunction with some of its optional features in your odbc.ini file, you could set the following connection string attributes:

```
Driver=ODBCHOME/lib/ivsqlsxx.so
Description=DataDirect SQL Server Wire Protocol
...
AlternateServers=(HostName=mySQLServer:PortNumber=1433:Database=Accounting,
HostName=255.201.11.24:PortNumber=1434:Database=Accounting)
...
ConnectionRetryCount=4
ConnectionRetryDelay=5
...
LoadBalancing=0
...
FailoverMode=1
...
FailoverPreconnect=1
...
```

Specifically, this odbc.ini configuration tells the driver to use two alternate servers as connection failover servers, to attempt to connect four additional times if the initial attempt fails, to wait five seconds between attempts, to try the primary and alternate servers in sequential order (do not use load balancing), to attempt reconnecting on new and lost connections, and to establish a connection with the primary and alternate servers at the same time.

Using security

The driver supports the following security features:

- *Authentication* is the process of identifying a user.
- *Data encryption* is the conversion of data into a form that cannot be easily understood by unauthorized users.

Authentication

On most computer systems, a password is used to prove a user's identity. This password often is transmitted

over the network and can possibly be intercepted by malicious hackers. Because this password is the one secret piece of information that identifies a user, anyone knowing a user's password can effectively be that user. Authentication methods protect the identity of the user.

The driver supports the following authentication methods:

- *User ID/password authentication* authenticates the user to the database using a database user name and password.
- *Client authentication* uses the user ID and password of the user logged onto the system on which the driver is running to authenticate the user to the database. The database server relies on the client to authenticate the user and does not provide additional authentication.
- *Kerberos authentication* is a trusted third-party authentication service that verifies user identities. The SQL Server Wire Protocol driver supports both Microsoft Entra Kerberos and MIT Kerberos implementations.
- *NTLM authentication* authenticates clients to the database through a challenge-response authentication mechanism that enables clients to prove their identities without sending a database password to the server. The driver supports NTLMv1 authentication on Windows clients.
- *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
- *Access token authentication* authenticates the user to the database using an access token and a pre-connect attribute.

Kerberos authentication

If you are using Kerberos, verify that your environment meets the requirements listed in the following table before you configure the driver for Kerberos authentication.

Table 4: Kerberos Authentication Requirements for the SQL Server Wire Protocol Driver

Component	Requirements
Database server	The database server must be administered by the same domain controller that administers the client.
Kerberos server	The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos KDC. Network authentication must be provided by Microsoft Entra Kerberos.
Client	The client must be administered by the same domain controller that administers the database server.

Kerberos authentication can take advantage of the user name and password maintained by the operating system to authenticate users to the database or use another set of user credentials specified by the application.

The Kerberos method requires knowledge of how to configure your Kerberos environment. This method supports both Microsoft Entra Kerberos and MIT Kerberos environments.

To use Kerberos authentication, the application user first must obtain a Kerberos Ticket Granting Ticket (TGT) from the Kerberos server. The Kerberos server verifies the identity of the user and controls access to services using the credentials contained in the TGT.

If the application uses Kerberos authentication from a UNIX/Linux client, the user must explicitly obtain a TGT. To obtain a TGT explicitly, the user must log onto the Kerberos server using the kinit command. For example, the following command requests a TGT from the server with a lifetime of 10 hours, which is renewable for 5 days:

```
kinit -l 10h -r 5d user
```

where *user* is the application user.

Refer to your Kerberos documentation for more information about using the kinit command and obtaining TGTs for users.

 If the application uses Kerberos authentication from a Windows client, the application user does not explicitly need to obtain a TGT. Microsoft Entra Kerberos automatically obtains a TGT for the user.

Related Links

For details, see the following topics:

- Connection string examples for configuring authentication
- odbc.ini file examples for configuring authentication

Connection string examples for configuring authentication

The following connection string configures the SQL Server Wire Protocol driver to use authentication, specifically Kerberos authentication. The examples contains the connection options necessary to configure Kerberos authentication as well as the minimum options required to establish a connection.

```
DSN=AcctSQLServer;HostName=AccountingSQLServer;AuthenticationMethod=4;
Database=Accounting;GSSClient=native;PortNumber=1433;UID=JohnSmith
```

odbc.ini file examples for configuring authentication

The following example odbc.ini file configures the 32-bit SQL Server Wire Protocol driver to use authentication, specifically Kerberos authentication. The examples contains the connection options necessary to configure Kerberos authentication as well as the minimum options required to establish a connection.

```
Driver=ODBCHOME/lib/ivsqlsxx.so
Description=DataDirect SQL Server Wire Protocol
...
AuthenticationMethod=4
...
Database=Accounting
...
GSSClient=native
```

```
...
HostName=AccountingSQLSServer
...
PortNumber=1433
...
UID=JohnSmith
...
```

Access token authentication

The driver supports the use of a pre-connection attribute that allows authentication with an access token obtained from a third-party application. The access token authentication method is available programmatically only.

To use the pre-connection `SQL_COPT_SS_ACCESS_TOKEN` attribute, set it to the `ACCESSTOKEN` pointer as follows:

```
typedef struct AccessToken
{
    SQLUINTEGER size;
    SQLCHAR data[];
} ACCESSTOKEN;
...
```

The access token comprises length of the token followed by bytes of opaque data. The token must be expanded with a `0` padding byte after each byte of data. The token must be in LittleEndian format and must not contain any NULL terminator. If an access token value is specified, then it will take precedence over other available authentication methods.

Note: Access tokens have a specified validity after generation. This can be configured on the server. If connected successfully, the session is maintained. If disconnected after the configured validity, then a new token is required to reconnect.

The following example uses the DSN-less format and includes the options for connecting with the access token flow.

```
#define SQL_COPT_SS_ACCESS_TOKEN 1256
...
SQLCHAR connString[]="Driver=DataDirect 8.0 SQL Server Wire Protocol;
HostName=myserver.database.windows.net;PortNumber=1234;Database=testdb";
SQLCHAR AccessToken="ey0jx12cd34efg5klm6543no32pqr10";
SQLUINTEGER dataSize=2*strlen(accessToken);
AccessToken*pAccToken=malloc(sizeof(ACCESSTOKEN)+dataSize);
pAccToken->size=dataSize;
for(int i=0,j=0;i<dataSize;i+=2,j++)
  {
    pAccToken->data[i]=accessToken[j];
    pAccToken->data[i+1]=0;
  }
SQLSetConnectAttr(hdbc,SQL_COPT_SS_ACCESS_TOKEN,(SQLPOINTER)pAccToken,SQL_IS_PO
INTER);
SQLDriverConnect(hdbc,NULL,connString,SQL_NTS,NULL,0,NULL,SQL_DRIVER_NOPROMPT);
```

```
free(pAccToken);
```

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 your Entra ID user name and password.
- Service principal user authentication: The driver authenticates using the object (principal) ID of the logical server and the client secret of your Azure 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.

When Entra ID authentication is enabled, all communications with the Azure SQL Database service are encrypted.

User and password authentication

To use user and password authentication with Entra ID:

- Specify values for minimum required options for establishing a connection:
  - Set the Host Name (`HostName`) option to specify either the IP address in IPv4 or IPv6 format, or the server name for your Azure server. For example, *`your_server`*`.database.windows.net`.
  - Set the Port Number (`PortNumber`) option to specify the TCP port of the primary database server that is listening for connections to the database.
  - Set Database (`Database`) option to specify the name of the database to which you want to connect.
  - If using data sources, set the Data Source Name (`DataSourceName`) to specify the name of your data source.
- Set the Authentication Method (`AuthenticationMethod`) option to `13` (Active Directory Password).
- Set the Trust Store (`Truststore`) connection option to specify the absolute path of the digital certificate file for the root CA certificates. The driver requires these certificates to maintain a secure connection.

  Note: For testing purposes, you can disable the truststore requirement by setting the Validate Server Certificate (`ValidateServerCertificate`) to `0` (disabled). Disabling the Validate Server Certificate option leaves your connection vulnerable to man-in-the-middle attacks; therefore, it is not recommended for extended use.
- Set the Host Name In Certificate (`HostNameInCertificate`) option to specify the host name for TLS/SSL certificate validation. For example, `*.database.windows.net`.
- Set the User Name (`LogonID`) option to specify your Entra ID username using the *`userid`*`@domain.com` format.
- Set the Password (`Password`) option to specify your Entra ID password.

For example, the following is a DSN-less connection string with only the required options for making a connection using Entra ID authentication with user name and password:

```
DRIVER={DataDirect 8.0 SQL Server Wire Protocol};AuthenticationMethod=13;
Database=MyDB;HostName=myserver.database.windows.net;
HostNameInCertificate=*.database.windows;PortNumber=1433;
TrustStore=\<truststore_path>\ca-bundle.crt;ValidateServerCertificate=1;
LogonID=test@domain.com;Password=secret;
```

The following example demonstrates a data source definition in the odbc.ini file with only the required options for making a connection using Entra ID authentication with user name and password:

```
[SQLServer Wire Protocol]
Driver=ODBCHOME/lib/ivsqls28.so
Description=DataDirect 8.0 SQL Server Wire Protocol
AuthenticationMethod=13
Database=MyDB
HostName=myserver.database.windows.net
HostNameInCertificate=*.database.windows
LogonID=test@domain.com
Password=secret
PortNumber=1433
TrustStore=/<truststore_path>/ca-bundle.crt
ValidateServerCertificate=1
```

Service principal user authentication

To use service principal user authentication with Entra ID:

- Specify values for minimum required options for establishing a connection:
  - Set the Host Name (`HostName`) option to specify either the IP address in IPv4 or IPv6 format, or the server name for your Azure server. For example, *`your_server`*`.database.windows.net`.
  - Set the Port Number (`PortNumber`) option to specify the TCP port of the primary database server that is listening for connections to the database.
  - Set the Database (`Database`) option to specify the name of the database to which you want to connect.
  - If using data sources, set the Data Source Name (`DataSourceName`) to specify the name of your data source.
- Set the Authentication Method (`AuthenticationMethod`) option to specify a value of `36` (Active Directory Service Principal).
- Set the Trust Store (`Truststore`) connection option to specify the absolute path of the digital certificate file for the root CA certificates. The driver requires these certificates to maintain a secure connection.
- Set the Host Name In Certificate (`HostNameInCertificate`) option to specify the host name for TLS/SSL certificate validation. For example, `*.database.windows.net`.
- Set the User Name (`LogonID`) option to specify the object (principal) ID of the Azure SQL logical server.
- Set the Password (`Password`) property to specify the client secret for your Entra ID application.

Note: For testing purposes, you can disable the Truststore and Host Name In Certificate requirement by setting the Validate Server Certificate (`ValidateServerCertificate`) to `0` (disabled). Disabling the Validate Server Certificate option leaves your connection vulnerable to man-in-the-middle attacks; therefore, it is not recommended for extended use.

For example, the following is a DSN-less connection string with only the required options for making a connection using principal ID authentication:

```
DRIVER={DataDirect 8.0 SQL Server Wire Protocol};AuthenticationMethod=36;
        Database=MyDB;HostName=myserver.database.windows.net;
        HostNameInCertificate=*.database.windows;PortNumber=1433;
        TrustStore=\<truststore_path>\ca-bundle.crt;
        LogonID=1234ABCD-1234-ABCD-1234-abcd1234ABCD1234;
        Password=ABcdEFg/hiJkLmNOPqR01stUvWxyzYx2wvUTsrQpO=;
```

The following example demonstrates a data source definition in the odbc.ini file with only the required options for making a connection using principal ID authentication:

```
[SQL Server Wire Protocol]
Driver=ODBCHOME/lib/ivsqls28.so
Description=DataDirect 8.0 SQL Server Wire Protocol
...
AuthenticationMethod=36
...
Database=MyDB
...
HostName=myserver.database.windows.net
...
HostNameInCertificate=*.database.windows
...
LogonID=1234ABCD-1234-ABCD-1234-abcd1234ABCD1234
...
Password=ABcdEFg/hiJkLmNOPqR01stUvWxyzYx2wvUTsrQpO=
...
PortNumber=1433
...
TrustStore=/<truststore_path>/ca-bundle.crt
...
```

Managed identity authentication

To use managed identity authentication with Entra ID:

- Specify values for minimum required options for establishing a connection:
  - Set the Host Name (`HostName`) option to specify either the IP address in IPv4 or IPv6 format, or the server name for your Azure server. For example, *your_server*`.database.windows.net`.
  - Set the Port Number (`PortNumber`) option to specify the TCP port of the primary database server that is listening for connections to the database.
  - Set the Database (`Database`) option to specify the name of the database to which you want to connect.

  - If using data sources, set the Data Source Name (`DataSourceName`) to specify the name of your data source.
- Set the Authentication Method (`AuthenticationMethod`) option to specify a value of `37` (Active Directory Managed Identity).
- Set the Trust Store (`Truststore`) option to specify the absolute path of the digital certificate file for the root CA certificates. The driver requires these certificates to maintain a secure connection.

  Note: For testing purposes, you can disable the truststore requirement by setting the Validate Server Certificate (`ValidateServerCertificate`) to `0` (disabled). Disabling the Validate Server Certificate option leaves your connection vulnerable to man-in-the-middle attacks; therefore, it is not recommended for extended use.

- Set the Host Name In Certificate (`HostNameInCertificate`) option to specify the host name for TLS/SSL certificate validation. For example, `*.database.windows.net`.
- (For user-assigned identities only) Set the User Name (`LogonID`) option to specify the client ID of your user-assigned managed identity used for authentication.

  Note: This value is required only if you have multiple user-assigned identities. If you have only one user-assigned identity, the driver will connect without specifying a value for this option.

For example, the following is a DSN-less connection string with only the required options for authenticating with a user-defined managed identity:

```
DRIVER={DataDirect 8.0 SQL Server Wire Protocol};AuthenticationMethod=37;
          Database=MyDB;HostName=myserver.database.windows.net;
          HostNameInCertificate=*.database.windows;PortNumber=1433;
          TrustStore=\<truststore_path>\ca-bundle.crt;
          LogonID=1234ABCD-1234-ABCD-1234-abcd1234ABCD1234;
```

The following example demonstrates a data source definition in the odbc.ini file with only the required options for authenticating with a user-defined managed identities:

```
[SQLServer Wire Protocol]
Driver=ODBCHOME/lib/ivsqls28.so
Description=DataDirect 8.0 SQL Server Wire Protocol
...
AuthenticationMethod=37
...
Database=MyDB
...
HostName=myserver.database.windows.net
...
HostNameInCertificate=*.database.windows
...
LogonID=1234ABCD-1234-ABCD-1234-abcd1234ABCD1234
...
PortNumber=1433
...
TrustStore=/<truststore_path>/ca-bundle.crt
...
```

Data encryption across the network

If your database connection is not configured to use data encryption, data is sent across the network in a format that is designed for fast transmission and can be decoded by interceptors, given some time and effort. For example, text data is often sent across the wire as clear text. Because this format does not provide complete protection from interceptors, you may want to use data encryption to provide a more secure transmission of data.

For example, you may want to use data encryption in the following scenarios:

- You have offices that share confidential information over an intranet.
- You send sensitive data, such as credit card numbers, over a database connection.
- You need to comply with government or industry privacy and security requirements.

Your Progress DataDirect *for* ODBC driver supports Transport Layer Security (TLS) and Secure Sockets Layer (SSL). TLS/SSL are industry-standard protocols for sending encrypted data over database connections. TLS/SSL secures the integrity of your data by encrypting information and providing client/server authentication.

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

TLS/SSL encryption

TLS/SSL works by allowing the client and server to send each other encrypted data that only they can decrypt. TLS/SSL negotiates the terms of the encryption in a sequence of events known as the *handshake*. During the handshake, the driver negotiates the highest TLS/SSL protocol available. The result of this negotiation determines the encryption cipher suite to be used for the TLS/SSL session.

The encryption cipher suite defines the type of encryption that is used for any data exchanged through a TLS/SSL connection. Some cipher suites are very secure and, therefore, require more time and resources to encrypt and decrypt data, while others provide less security, but are also less resource intensive.

The handshake involves the following types of authentication:

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

Refer to "SSL encryption cipher suites" in the *Progress DataDirect for ODBC Drivers Reference* for a list of the encryption cipher suites supported by the drivers.

Certificates

TLS/SSL encryption requires the use of a digitally-signed document, an x.509 standard certificate, for authentication and the secure exchange of data. The purpose of this certificate is to tie the public key contained in the certificate securely to the person/company that holds the corresponding private key. Your Progress DataDirect *for* ODBC drivers supports many popular formats. Supported formats include:

- DER Encoded Binary X.509
- Base64 Encoded X.509
- PKCS #12 / Personal Information Exchange

TLS/SSL server authentication

When the client makes a connection request, the server presents its certificate for the client to accept or deny. The client checks the issuer of the certificate against a list of trusted Certificate Authorities (CAs) whose root certificates reside in one or both of the following stores on the client:

- On Windows operating systems: A permanent storage known as *Windows certificate store*. To learn how to import the required root certificates into the Windows certificate store, see "Importing root certificates into the Windows certificate store."
- On both Windows and non-Windows operating systems: An encrypted file known as *truststore file*. Most truststore files are password-protected. The driver must be able to locate the truststore file and unlock it with the appropriate password. Two connection options are available to the driver to provide this information: Trust Store (Truststore) and Trust Store Password (TruststorePassword).

If the server certificate matches a root certificate in either of the stores, an encrypted connection is established between the client and the server. If the certificate does not match, the connection fails and the client generates an error.

Alternatively, you can configure the driver to trust any certificate sent by the server, even if the issuer is not a trusted CA. Allowing a driver to trust any certificate sent from the server is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment. Setting the Validate Server Certificate (ValidateServerCertificate) connection option to false allows the driver to accept any certificate returned from the server regardless of whether the issuer of the certificate is a trusted CA.

To configure the driver to use data encryption via TLS/SSL server authentication:

- Set the minimum required options for all connections:
  - Set the Host Name (HostName) option to specify the name or the IP address of the server to which you want to connect.
  - Set the Port Number (PortNumber) option to specify the port number of the server listener. The default is `1433`.
  - Set the Database Name (Database) option to specify the name of the database to which you want to connect.
- Set the Encryption Method (EncryptionMethod) option to one of the following valid values:
  - Set to `1` (SSL) to specify that data is encrypted using the TLS/SSL protocols specified in the Crypto Protocol Version connection option.
  - Set to `6` (RequestSSL) to specify that the login request and data are encrypted using TLS/SSL if the server is configured for TLS/SSL encryption. If the server is not configured for TLS/SSL, an unencrypted connection is established. The TLS/SSL protocol used is determined by the setting of the Crypto Protocol Version connection option.
  - Set to `7` (LoginSSL) to specify that the login request is encrypted using TLS/SSL regardless of whether the server is configured for TLS/SSL. The data is encrypted using TLS/SSL if the server is configured for TLS/SSL, and the data is unencrypted if the server is not configured for TLS/SSL. The TLS/SSL

protocol used is determined by the setting of the Crypto Protocol Version connection option.

- Set to `8` (Strict) to configure the driver to use 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=1) for the connection, regardless of the setting of the Validate Server Certificate option.
  - You must specify a truststore containing the server certificate against which the server will be validated at connection.

- Set the Validate Server Certificate (ValidateServerCertificate) option to determine whether the driver validates the certificates sent by the server. When it is set to `1`, the driver validates the certificates. When it is set to `0`, the driver does not validate the certificates.
- Set the Host Name In Certificate (HostNameInCertificate) option to specify the host name that is specified in the Subject of the certificate. This option 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. Consult your TLS/SSL administrator for the correct value.
- Set the Trust Store (Truststore) option to specify either the full path of the truststore file or the contents of the TLS/SSL certificates.

  Note: To allow the client to use TLS/SSL server authentication without storing the truststore file on the disk, you can specify the contents of the root certificates using the Trust Store connection option. Alternatively, you can use the pre-connection attribute, `SQL_COPT_INMEMORY_TRUSTSTORECERT`, to specify the certificate content. For more information, see "Trust Store" and "Using SQL_COPT_INMEMORY_TRUSTSTORECERT".

- Set the Truststore Password (TruststorePassword) option to specify the password that is used to access the truststore file.
- Optionally, set the Crypto Protocol Version (CryptoProtocolVersion) to specify the cryptographic protocol versions supported by your database server that you want to allow your driver to use. For example, `CryptoProtocolVersion=TLSv1.3,TLSv1.2`.

  Note: TLSv1.3 is currently supported only when strict connection encryption is enabled (`EncryptionMethod=8`).

- Optionally, set the Enable FIPS (EnableFIPS) connection option to `1` to allow the driver to load the FIPS provider. The FIPS provider, introduced in OpenSSL 3.0, contains a set of approved cryptographic algorithms that conform to the Federal Information Processing Standards (FIPS) specified in FIPS 140-2. If you do not specify a value for Enable FIPS, the driver uses its default value (`0`) and loads the default provider.

  Note:
  - The FIPS provider is supported only on the following platforms: Windows 64-bit, Linux 64-bit, and AIX 64-bit.
  - Do not set the Truststore Password connection option when using the FIPS provider. The truststore password uses the PKCS12KDF algorithm, which is not an approved FIPS algorithm. Hence, it must not be specified when using the FIPS provider.
  - For using the FIPS and default providers, the certificates must be generated using the OpenSSL

3.0-compliant cryptographic algorithms. See "Generating TLS/SSL certificates using OpenSSL 3.0-compliant algorithms" for more information.

The following examples show how to configure the driver to establish a connection via user ID/password authentication and use data encryption via TLS/SSL server authentication. In these examples, since `ValidateServerCertificate=1` and `EnableFIPS=1`, the driver validates the certificate sent by the server and the host name specified by the HostNameInCertificate option, and loads the FIPS provider for data encryption.

Connection string

Truststore:

```
DRIVER=DataDirect 8.0 SQL Server Wire Protocol;EnableFIPS=1;
EncryptionMethod=1;HostName=YourServer;HostNameInCertificate=MySubjectAltName;
PortNumber=1433;Database=SQLSdb1;Truststore=TrustStoreName;
ValidateServerCertificate=1
```

Note: On Windows, the driver validates the server certificate against the root certificates available in both truststore and Windows certificate store. If a matching certificate is found in either of the stores, the connection is established.

Windows certificate store:

```
DRIVER=DataDirect 8.0 SQL Server Wire Protocol;EnableFIPS=1;
EncryptionMethod=1;HostName=YourServer;HostNameInCertificate=MySubjectAltName;
PortNumber=1433;Database=SQLSdb1;ValidateServerCertificate=1
```

Note: The LogonID and Password options are not required to be stored in the connection string. They can also be sent separately by the application using the SQLConnect ODBC API. For SQLDriverConnect and SQLBrowseConnect, they will need to be specified in the connection string.

odbc.ini

Truststore:

```
Driver=ODBCHOME/lib/ivsqlsxx.so
Description=DataDirect SQL Server Wire Protocol
...
EnableFIPS=1
...
EncryptionMethod=1
...
HostName=YourServer
...
HostNameInCertificate=MySubjectAltName
...
PortNumber=1433
```

```
...
Database=SQLSdb1
...
Truststore=TrustStoreName
...
ValidateServerCertificate=1
...
```

Note: On Windows, the driver validates the server certificate against the root certificates available in both truststore and Windows certificate store. If a matching certificate is found in either of the stores, the connection is established.

Windows certificate store:

```
Driver=ODBCHOME/lib/ivsqlsxx.so
Description=DataDirect SQL Server Wire Protocol
...
EnableFIPS=1
...
EncryptionMethod=1
...
HostName=YourServer
...
HostNameInCertificate=MySubjectAltName
...
PortNumber=1433
...
Database=SQLSdb1
...
ValidateServerCertificate=1
...
```

Note: The LogonID and Password options are not required to be stored in the data source. They can also be sent separately by the application using the SQLConnect ODBC API. For SQLDriverConnect and SQLBrowseConnect, they will need to be specified in the data source or connection string.

Related Links

For details, see the following topics:

- Using SQL_COPT_INMEMORY_TRUSTSTORECERT
- Importing root certificates into the Windows certificate store

Using SQL_COPT_INMEMORY_TRUSTSTORECERT

`SQL_COPT_INMEMORY_TRUSTSTORECERT` is a pre-connection attribute that specifies the contents of the TLS/SSL certificates for TLS/SSL server authentication. When using `SQL_COPT_INMEMORY_TRUSTSTORECERT`, the driver stores the certificate content in memory, which eliminates the need to store the truststore file on the disk and lets applications use TLS/SSL server authentication without any disk dependency.

Note: The certificate content can be specified using the Trust Store (Truststore) connection option as well. However, if it is specified using both Trust Store and `SQL_COPT_INMEMORY_TRUSTSTORECERT`, `SQL_COPT_INMEMORY_TRUSTSTORECERT` takes precedence over Trust Store.

The following example shows how to specify the contents of 3 certificates using `SQL_COPT_INMEMORY_TRUSTSTORECERT`:

```
SQLCHAR certificate[] = "
-----BEGIN CERTIFICATE-----12345abc-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----abcd123456-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----aabbcc11----END CERTIFICATE-----";
//The content of each certificate must be specified between -----BEGIN
CERTIFICATE----- and -----END CERTIFICATE-----. Also, the number of dashes
(-----) must be the same before and after both BEGIN CERTIFICATE and END
CERTIFICATE.

...

SQLSetConnectAttr(dbc, SQL_COPT_INMEMORY_TRUSTSTORECERT,
(SQLPOINTER)certificate, SQL_IS_POINTER);

ret = SQLDriverConnect(dbc, NULL,
(SQLCHAR*)"DSN=SQL ServerWP_SSL;UID=jsmith;PWD=secret", SQL_NTS,
NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
```

Importing root certificates into the Windows certificate store

This section provides you with an overview of the steps required to import the required root certificates from a truststore file to the Windows certificate store.

You can import root certificates using either the Certificate Import Wizard or a PowerShell script.

Related Links

For details, see the following topics:

- Importing root certificates using Certificate Import Wizard
- Importing root certificates using a PowerShell script

To import root certificates using Certificate Import Wizard:

1. Double-click the trustore file. The **Certificate Import Wizard** window appears.
2. Select the **Current User** radio button; then, click **Next**.
3. Verify the file path and name available in the **File name** field; then, click **Next**.
4. Enter the password to unlock the truststore file; then, click **Next**.
5. Select the **Automatically select the certificate store based on the type of certificate** radio button; then, click **Next**.
6. Click **Finish**.

The root certificates are imported into the following location in the Windows certificate store: **Certificates** > **Trusted Root Certification Authorities** > **Certificates**.

Note: At times, Windows doesn't trust the imported certificates and imports them into **Certificates** > **Intermediate Certificate Authorities** > **Certificates**. In such cases, manually copy the imported certificates from **Intermediate Certificate Authorities** to **Trusted Root Certification Authorities**.

To import root certificates using a PowerShell script:

1. Open PowerShell in Administrator mode.
2. Type the following command; then, press **ENTER**.

```
Import-PfxCertificate -Password (ConvertTo-SecureString -String
"truststore_password" -AsPlainText -Force) -CertStoreLocation
Cert:\LocalMachine\Root -FilePath truststore_filepath
```

where:

truststore_password

is the password that is used to access the truststore file.

truststore_filepath

is the path to the directory where the truststore file is located.

The root certificates are imported into the following location in the Windows certificate store: **Certificates** > **Trusted Root Certification Authorities** > **Certificates**.

Note: At times, Windows doesn't trust the imported certificates and imports them into **Certificates** > **Intermediate Certificate Authorities** > **Certificates**. In such cases, manually copy the imported certificates from **Intermediate Certificate Authorities** to **Trusted Root Certification Authorities**.

TLS/SSL client authentication

If the server is configured for TLS/SSL client authentication, the server asks the client to verify its identity after the server identity has been proven. Similar to server authentication, the client sends a public certificate to the server to accept or deny. The client stores its public certificate in an encrypted file known as a *keystore*. Public certificates are paired with a private key in the keystore. To send the public certificate, the driver must access the private key.

Like the truststore, most keystores are password-protected. The driver must be able to locate the keystore and unlock the keystore with the appropriate password. Two connection options are available to the driver to provide this information: Keystore (KeyStore) and Keystore Password (KeyStorePassword). The value of KeyStore is a pathname that specifies the location of the keystore file. The value of Keystore Password is the password required to access the keystore.

The private keys stored in a keystore can be individually password-protected. In many cases, the same password is used for access to both the keystore and to the individual keys in the keystore. It is possible, however, that the individual keys are protected by passwords different from the keystore password. The driver needs to know the password for an individual key to be able to retrieve it from the keystore. An additional connection option, Key Password (KeyPassword), allows you to specify a password for an

individual key.

To configure the driver to use data encryption via TLS/SSL client authentication:

- Set the minimum required options for all connections:
  - Set the Host Name (HostName) option to specify the name or the IP address of the server to which you want to connect.
  - Set the Port Number (PortNumber) option to specify the port number of the server listener. The default is `1433`.
  - Set the Database Name (Database) option to specify the name of the database to which you want to connect.
- Set the Encryption Method (EncryptionMethod) option to one of the following to valid values:
  - If set to `1` (SSL), data is encrypted using the TLS/SSL protocols specified in the Crypto Protocol Version connection option.
  - Set to `6` (RequestSSL) to specify the login request and data are encrypted using TLS/SSL if the server is configured for TLS/SSL encryption. If the server is not configured for TLS/SSL, an unencrypted connection is established. The TLS/SSL protocol used is determined by the setting of the Crypto Protocol Version connection option.
  - Set to `7` (LoginSSL) to specify that the login request is encrypted using TLS/SSL regardless of whether the server is configured for TLS/SSL. The data is encrypted using TLS/SSL if the server is configured for TLS/SSL, and the data is unencrypted if the server is not configured for TLS/SSL. The TLS/SSL protocol used is determined by the setting of the Crypto Protocol Version connection option.
  - Set to `8` (Strict) to configure the driver to use 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=1) for the connection, regardless of the setting of the Validate Server Certificate option.
    - You must specify a truststore containing the server certificate against which the server will be validated at connection.

- Set the Validate Server Certificate (ValidateServerCertificate) option to determine whether the driver validates the certificates sent by the server. When it is set to `1`, the driver validates the certificates. When it is set to `0`, the driver does not validate the certificates.
- Set the Host Name In Certificate (HostNameInCertificate) option to specify the host name that is specified in the Subject of the certificate. This option 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. Consult your TLS/SSL administrator for the correct value.
- Set the Key Store (Keystore) option to specify the location of the keystore file.
- Set the Keystore Password (KeystorePassword) option to specify the password that is used to access the keystore file.
- Optionally, set the Crypto Protocol Version (CryptoProtocolVersion) to specify the cryptographic protocol versions supported by your database server that you want to allow your driver to use. For example, `CryptoProtocolVersion=TLSv1.3,TLSv1.2`.

  Note: TLSv1.3 is currently supported only when strict connection encryption is enabled

  ---

(EncryptionMethod=8).

- Optionally, set the Enable FIPS (EnableFIPS) connection option to 1 to allow the driver to load the FIPS provider. The FIPS provider, introduced in OpenSSL 3.0, contains a set of approved cryptographic algorithms that conform to the Federal Information Processing Standards (FIPS) specified in FIPS 140-2. If you do not specify a value for Enable FIPS, the driver uses its default value (0) and loads the default provider.

  Note:
  - The FIPS provider is supported only on the following platforms: Windows 64-bit, Linux 64-bit, and AIX 64-bit.
  - For using the FIPS and default providers, the certificates must be generated using the OpenSSL 3.0-compliant cryptographic algorithms. See "Generating TLS/SSL certificates using OpenSSL 3.0-compliant algorithms" for more information.
  - Do not set the Keystore Password connection option when using the FIPS provider. The keystore password uses the PKCS12KDF algorithm, which is not an approved FIPS algorithm. Hence, it must not be specified when using the FIPS provider.

The following examples show how to configure the driver to establish a connection via user ID/password authentication and use data encryption via TLS/SSL client authentication. In these examples, since ValidateServerCertificate=1 and EnableFIPS=1, the driver validates the certificate sent by the server and the host name specified by HostNameInCertificate, and loads the FIPS provider for data encryption.

Connection string

```
DRIVER=DataDirect 8.0 SQL Server Wire Protocol;EnableFIPS=1;
EncryptionMethod=1;HostName=YourServer;HostNameInCertificate=MySubjectAltName;
PortNumber=1433;Database=Payroll;Keystore=KeyStoreName;ValidateServerCertificat
e=1;
```

Note: The LogonID and Password options are not required to be stored in the connection string. They can also be sent separately by the application using the SQLConnect ODBC API. For SQLDriverConnect and SQLBrowseConnect, they will need to be specified in the connection string.

odbc.ini

```
Driver=ODBCHOME/lib/ivsqlsxx.so
Description=DataDirect SQL Server Wire Protocol
...
EnableFIPS=1
...
EncryptionMethod=1
...
HostName=YourServer
...
HostNameInCertificate=MySubjectAltName
...
```

```
PortNumber=1433
...
Database=Payroll;
...
KeyStore=KeyStoreName
...
ValidateServerCertificate=1
...
```

Note: The LogonID and Password options are not required to be stored in the data source. They can also be sent separately by the application using the SQLConnect ODBC API. For SQLDriverConnect and SQLBrowseConnect, they will need to be specified in the data source or connection string.

Generating TLS/SSL certificates using OpenSSL 3.0-compliant algorithms

For using the OpenSSL 3.0 providers (FIPS and default), the certificates for TLS/SSL encryption must be generated using the OpenSSL 3.0-compliant cryptographic algorithms.

There are multiple ways of generating these certificates. The following commands demonstrate one of them. You can use these commands to generate the certificates and add them to the truststore and keystore files.

Note: The openssl.exe file is required for running these commands. You can download it from the official OpenSSL website.

Truststore:

```
openssl.exe pkcs12 -in certificate_name -export -out truststore_filename
-nokeys -keypbe cryptographic_algorithm -certpbe cryptographic_algorithm
-password pass:truststore_password -nomac
```

where:

`certificate_name`

is the name of the certificate you are generating.

`truststore_filename`

is the name of the truststore file.

`cryptographic_algorithm`

is the cryptographic algorithm you are using to generate the certificate.

`truststore_password`

is the password required for accessing the truststore file.

Example:

```
openssl.exe pkcs12 -in nc-thunder-SHA256.cer -export -out truststorepw.pfx
-nokeys -keypbe AES-256-CBC -certpbe AES-256-CBC -password pass:MyPassW0rd
-nomac
```

Keystore:

```
openssl.exe pkcs12 -in certificate_name -inkey privatekey_file -export -out
keystore_file -keypbe cryptographic_algorithm -certpbe cryptographic_algorithm
-nomac
```

where:

certificate_name

is the name of the certificate you are generating.

privatekey_file

is the name of the file that contains the private key.

truststore_filename

is the name of the keystore file.

cryptographic_algorithm

is the cryptographic algorithm you are using to generate the certificate.

Example:

```
openssl.exe pkcs12 -in nc-thunder-SHA256.cer -inkey ./file.pem -export -out
keystorepw.pfx -keypbe AES-256-CBC -certpbe AES-256-CBC -nomac
```

Note: If you are using the Windows certificate store for TLS/SSL encryption, import the certificates generated with the OpenSSL 3.0-compliant algorithms into the store.

Designating an OpenSSL Library

Important: Currently, the driver supports only version 3.0 of the OpenSSL library.

The driver uses OpenSSL library files (TLS/SSL Support Files) to implement cryptographic functions for data sources or connections when encrypting data. By default, the driver is configured to use the most secure version of the library installed with the product; however, you can designate a different version to address security vulnerabilities or incompatibility issues with your current library. Although the driver is only certified against libraries provided by Progress, you can also designate libraries that you supply. The methods described in this section can be used to designate an OpenSSL library file.

Note: For the default library setting, current information, and a complete list of installed OpenSSL libraries, refer to the readme file installed with your product.

File replacement

In the default configuration, the drivers use the OpenSSL library file located in the \drivers subdirectory for Windows installations and the /lib subdirectory for UNIX/Linux. You can replace this file with a different library to change the version used by the drivers. When using this method, the replacement file must contain both the cryptographic and TLS/SSL libraries and use the same file name as the default library. For example, the latest version of the library files use the following naming conventions:

Windows:

- Version 3.0: *xx*openssl30.dll

UNIX/Linux:

- Version 3.0: *xx*openssl30.so [.sl]

Designating the absolute path to a library

For libraries that do not use the default directory structure or file names, you must specify the absolute path to your cryptographic library for the CryptoLibName (CryptoLibName) option and the absolute path to your TLS/SSL library for the SSLLibName (SSLLibName) option. If you are using OpenSSL library files provided by Progress, these libraries are combined into a single file; therefore, the value specified for these options should be the same. For non-Progress library files, the libraries may use separate files, which would require specifying the unique paths to the libeay32.dll (cryptographic library) and ssleay32.dll (TLS/SSL library) files.

If you are using a GUI, these options are not exposed on the setup dialog. Instead, use the Extended Options field on the Advanced tab to configure these options. See "CryptoLibName" and "SSLLibName" for details.

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.

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 following steps 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. See the following "Using keystore providers" section for details.

   Note: You can dictate whether column encryption keys are persisted in the cache using the Key Cache Time To Live (AEKeyCacheTTL) option. See "Caching column encryption keys" for more information.
4. The driver encrypts the parameters using the unencrypted 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.

Enabling Always Encrypted

You can configure the default behavior for Always Encrypted by specifying one of following values for the Column Encryption connection option (ColumnEncryption):

- 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`, 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` (default), 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.

The behavior specified for the Column Encryption option acts as the default for the connection; however, you can override this behavior on a per-statement basis by specifying a value for the `SQL_SOPT_SS_COLUMN_ENCRYPTION` statement attribute. By using this statement attribute, you can toggle support for Always Encrypted to suit the applications requirements without having to establish a new connection. This allows you to avoid some of the overhead associated with encrypting and decrypting data when accessing tables that do not contain encrypted columns. To use the `SQL_SOPT_SS_COLUMN_ENCRYPTION` statement attribute, the Column Encryption connection option must be set to `Enabled` or `ResultsetOnly`.

For details on configuring the Column Encryption option, see "Column Encryption." See the following section for more information on using the statement attribute.

Depending on your keystore provider, you may need to further configure the driver to connect when Always

Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`). See "Using keystore providers" for details.

Enabling Always Encrypted programmatically

To override the default behavior for the connection or DSN when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`), specify one of the following values for the `SQL_SOPT_SS_COLUMN_ENCRYPTION` statement attribute in a SQL Statement:

Table 5: SQL_SOPT_SS_COLUMN_ENCRYPTION attribute values

Values	Behavior
`SQL_CE_ENABLED` (`1`)	Always Encrypted functionality is fully enabled. 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.
`SQL_CE_RESULTSETONLY` (`2`)	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.
`SQL_CE_DISABLED` (`0`)	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. Execution of statements containing parameters that affect encrypted columns are not supported and will return an error.

Using keystore providers

Keystore providers store the column master keys used for decrypting column encryption keys when using Always Encrypted. The driver currently supports the Windows Certificate Store, Azure Key Vault, and PKCS #12 File providers. If the location of a PKCS #12 file is specified, the driver uses the PKCS #12 file located at the specified location as a provider. Otherwise, the driver dynamically determines which provider to use for encrypting parameters or decrypting data in result sets based on the encryption metadata received from the server.

Windows Certificate Store

The Windows Certificate Store is a local repository of certificates available only on Windows platforms. When using this provider, the column master key is stored locally on the client machine, which reduces the need to make calls over a network. The driver does not require any additional configuration to use the Windows Certificate Store as a provider when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`).

Azure Key Vault

The Azure Key Vault is a certificate repository hosted on Azure platforms. This provider offers several advantages over the Windows Certificate Store, including the ability to access keys when the application is running on any platform. In addition, keys do not need to be copied to and cached on the local machine. However, unless the application is running on Azure, calls to the key vault must be made over a WAN, which can affect performance. To access the column master key, the principal ID and Client Secret must be used to authenticate against the Azure Key Vault. You can specify the principal ID and Client Secret using the

following options:

- Key Store Principal Id (AEKeystorePrincipalId): Specifies the principal ID used to authenticate against the Azure Key Vault. See "Key Store Principal Id" for a detailed description.
- Key Store Secret (AEKeystoreClientSecret): Specifies the Client Secret used to authenticate against the Azure Key Vault. See "Key Store Secret" for a detailed description.

PKCS #12 File

The driver supports storing the column master key in a PKCS #12 file. The PKCS #12 file can be stored locally on the disk on any platform the driver supports. To allow the driver to access this provider, values for the following options must be specified:

- AE Keystore Location (AEKeyStoreLocation): Specifies the absolute path to the PKCS #12 file.
- AE Keystore Secret (AEKeyStoreSecret): Specifies the password used to access the PKCS #12 file. This option is needed only when the PKCS #12 file is password-protected.

Caching column encryption keys

Caching column encryption keys improves performance by eliminating 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 at the end of a connection; however, depending on the security needs of your environment, you may not want to store keys in the cache at all. You can determine whether the driver caches column encryption keys by specifying the following values for the Key Cache Time To Live (AEKeyCacheTTL) option:

- If set to `-1`, the driver caches column encryption keys on a per connection basis. The keys remain in the cache until the connection is closed or the application exits.
- If set to `0`, the driver does not cache column encryption keys.

By default, the driver caches column encryption keys on a per connection basis (`AEKeyCacheTTL=-1`). The driver caches keys only when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`). See "Key Cache Time To Live" for details.

Connection string examples

The following connection string configures the driver to use Always Encrypted with the Windows Certificate Store:

```
DRIVER={DataDirect 8.0 SQL Server Wire
Protocol};ColumnEncryption=Enabled;HostName=YourServer;PortNumber=1433;
```

The following connection string configures the driver to use Always Encrypted with the Azure Key Vault. Unlike connections using Windows Certificate Store, using Azure Key Vault requires the Key Store Principal Id (`AEKeystorePrincipalId`) and Key Store Secret (`AEKeyStoreClientSecret`) connection options.

```
DRIVER={DataDirect 8.0 SQL Server Wire
Protocol};AEKeystorePrincipalId=789a8b7c-6d5e-432f-1gh2-3ijk45678987;
AEKeyStoreClientSecret=ABcdEFg/
hiJkLmNOPqR01tuVwXwyzYw2xwVUtsRQ=;ColumnEncryption=Enabled;
HostName=YourServer;PortNumber=1433;
```

See "Configuring and connecting to data sources" for more information on configuring and connecting to data sources.

Note: The keystore provider used by the driver is based on the encryption metadata received from the server. Specifying values for the Key Store Principal Id and Key Store Secret connections does not determine that the driver uses the Azure Key Vault.

The following connection string configures the driver to use Always Encrypted with a PKCS #12 file. To use a PKCS #12 file, the driver requires values for the AE Keystore Location (`AEKeyStoreLocation`) and AE Keystore Secret (`AEKeyStoreSecret`) connection options. The value for the AE Keystore Secret is needed only when the PKCS #12 file is password-protected.

```
DRIVER={DataDirect 8.0 SQL Server Wire Protocol};AEKeyStoreLocation=C:\abc\
keystore.pfx;
AEKeyStoreSecret=ABcdEFg/
hiJkLmNOPqR01tuVwXwyzYw2xwVUtsRQ=;ColumnEncryption=Enabled;
HostName=YourServer;PortNumber=1433;
```

Using DataDirect Connection Pooling

Supported on Windows, UNIX, and Linux only.

The SQL Server Wire Protocol driver supports DataDirect Connection Pooling on Windows, UNIX, and Linux platforms. Connection pooling allows you to *reuse* connections rather than creating a new one every time the driver needs to establish a connection to the underlying database. The driver enables connection pooling without requiring changes to your client application.

Note: Connection pooling works only with connections that are established using SQLConnect or SQLDriverConnect with the SQL_DRIVER_NO_PROMPT argument and only with applications that are thread-enabled.

DataDirect connection pooling that is implemented by the DataDirect driver is different than connection pooling implemented by the Windows Driver Manager. The Windows Driver Manager opens connections dynamically, up to the limits of memory and server resources. DataDirect connection pooling, however, allows you to control the number of connections in a pool through the Min Pool Size (minimum number of connections in a pool) and Max Pool Size (maximum number of connections in a pool) connection options. In addition, DataDirect connection pooling is cross-platform, allowing it to operate on UNIX and Linux. See "Summary of pooling-related options" for details about how the connection options manage DataDirect connection pooling.

Important: On Windows, do not use connection pooling for the Windows Driver Manager at the same time as DataDirect connection pooling.

Creating a connection pool

Each connection pool is associated with a specific connection string. By default, the connection pool is created when the first connection with a unique connection string connects to the data source. The pool is populated with connections up to the minimum pool size before the first connection is returned. Additional connections can be added until the pool reaches the maximum pool size. If the Max Pool Size option is set to `10` and all connections are active, a request for an eleventh connection has to wait in queue for one of the 10 pool connections to become idle. The pool remains active until the process ends or the driver is unloaded.

If a new connection is opened and the connection string does not exactly match an existing pool, a new pool must be created. By using the same connection string, you can enhance the performance and scalability of your application.

Adding connections to a pool

A connection pool is created in the process of creating each unique connection string that an application uses. When a pool is created, it is populated with enough connections to satisfy the minimum pool size requirement, set by the Min Pool Size connection option. The maximum pool size is set by the Max Pool Size connection option. If an application needs more connections than the number set by Min Pool Size, the driver allocates additional connections to the pool until the number of connections reaches the value set by Max Pool Size.

Once the maximum pool size has been reached and no usable connection is available to satisfy a connection request, the request is queued in the driver. The driver waits for the length of time specified in the Login Timeout connection option for a usable connection to return to the application. If this time period expires and a connection has not become available, the driver returns an error to the application.

A connection is returned to the pool when the application calls SQLDisconnect. Your application is still responsible for freeing the handle, but this does not result in the database session ending.

Removing connections from a pool

A connection is removed from a connection pool when it exceeds its lifetime as determined by the Load Balance Timeout connection option. In addition, DataDirect has created connection attributes described in the following table to give your application the ability to reset connection pools. If connections are in use at the time of these calls, they are marked appropriately. When SQLDisconnect is called, the connections are discarded instead of being returned to the pool.

Table 6: Pool Reset Connection Attributes

Connection Attribute	Description
SQL_ATTR_CLEAR_POOLS Value: SQL_CLEAR_ALL_CONN_POOL	Calling SQLSetConnectAttr (SQL_ATTR_CLEAR_POOLS, SQL_CLEAR_ALL_CONN_POOL) clears all the connection pools associated with the driver that created the connection.This is a write-only connection attribute. The driver returns an error if SQLGetConnectAttr (SQL_ATTR_CLEAR_POOLS) is called.
SQL_ATTR_CLEAR_POOLS Value: SQL_CLEAR_CURRENT_CONN_POOL	Calling SQLSetConnectAttr (SQL_ATTR_CLEAR_POOLS, SQL_CLEAR_CURRENT_CONN_POOL) clears the connection

Connection Attribute	Description
	pool that is associated with the current connection.This is a write-only connection attribute. The driver returns an error if SQLGetConnectAttr (SQL_ATTR_CLEAR_POOLS) is called.

Note: By default, if removing a connection causes the number of connections to drop below the number specified in the Min Pool Size option, a new connection is not created until an application needs one.

Handling dead connections in a pool

What happens when an idle connection loses its physical connection to the database? For example, suppose the database server is rebooted or the network experiences a temporary interruption. An application that attempts to connect could receive errors because the physical connection to the database has been lost.

Your Progress DataDirect *for* ODBC driver handles this situation transparently to the user. The application does not receive any errors on the connection attempt because the driver simply returns a connection from a connection pool. The first time the connection handle is used to execute a SQL statement, the driver detects that the physical connection to the server has been lost and attempts to reconnect to the server *before* executing the SQL statement. If the driver can reconnect to the server, the result of the SQL execution is returned to the application; no errors are returned to the application.

The driver uses connection failover option values, if they are enabled, when attempting this seamless reconnection; however, it attempts to reconnect even if these options are not enabled. See "Connection failover" for information about configuring the driver to connect to a backup server when the primary server is not available.

Note: If the driver cannot reconnect to the server (for example, because the server is still down), an error is returned indicating that the reconnect attempt failed, along with specifics about the reason the connection failed.

The technique that Progress DataDirect uses for handling dead connections in connection pools allows for maximum performance of the connection pooling mechanism. Some drivers periodically test the server with a dummy SQL statement while the connections sit idle. Other drivers test the server when the application requests the use of the connection from the connection pool. Both of these approaches add round trips to the database server and ultimately slow down the application during normal operation.

Connection pool statistics

Progress DataDirect has created a connection attribute to monitor the status of the Progress DataDirect for ODBC connection pools. This attribute, which is described in the following table, allows your application to fetch statistics for the pool to which a connection belongs.

Table 7: Pool Statistics Connection Attribute

Connection Attribute	Description
SQL_ATTR_POOL_INFO Value: SQL_GET_POOL_INFO	Calling SQLGetConnectAttr (SQL_ATTR_POOL_INF, SQL_GET_POOL_INFO) returns a PoolInfoStruct that contains the

Connection Attribute	Description
	statistics for the connection pool to which this connection belongs. This PoolInfoStruct is defined in qesqlext.h. For example: `SQLGetConnectAttr(hdbc, SQL_ATTR_POOL_INFO, PoolInfoStruct *, SQL_LEN_BINARY_ATTR(PoolInfoStruct), &len);` This is a read-only connection attribute. The driver returns an error if SQLSetConnectAttr (SQL_ATTR_POOL_INFO) is called.

Summary of pooling-related options

The following table summarizes how connection pooling-related connection options work with the drivers. See "Connection option descriptions" for additional details about configuring the options.

Table 8: Summary: Connection Pooling Connection Options

Option	Characteristic
Connection Pooling (Pooling)	Specifies whether to use the driver's connection pooling. If set to `1` (Enabled), the driver uses connection pooling. If set to `0` (Disabled), the driver does not use connection pooling. **Default:** `0` (Disabled)
Connection Reset (ConnectionReset)	Determines whether the state of connections that are removed from the connection pool for reuse by the application is reset to the initial configuration of the connection.If set to `1` (Enabled), the state of connections removed from the connection pool for reuse by an application is reset to the initial configuration of the connection. Resetting the state can negatively impact performance because additional commands must be sent over the network to the server to reset the state of the connection. If `0` (Disabled), the state of connections is not reset. **Default:** `0` (Disabled)
Load Balance Timeout (LoadBalanceTimeout)	An integer value to specify the amount of time, in seconds, to keep connections open in a connection pool. If set to 0, inactive connections are kept open. If set to `x`, inactive connections are closed after the specified number of seconds passes. Default: `0`

Option	Characteristic
Max Pool Size (MaxPoolSize)	An integer value to specify the maximum number of connections within a single pool. **Default:** `100`
Min Pool Size (MinPoolSize)	An integer value to specify the minimum number of connections that are opened and placed in a connection pool when it is created. If set to `0`, no connections are opened in addition to the current existing connection. **Default:** `0`

Using DataDirect Bulk Load

Supported on Windows, UNIX, and Linux only.

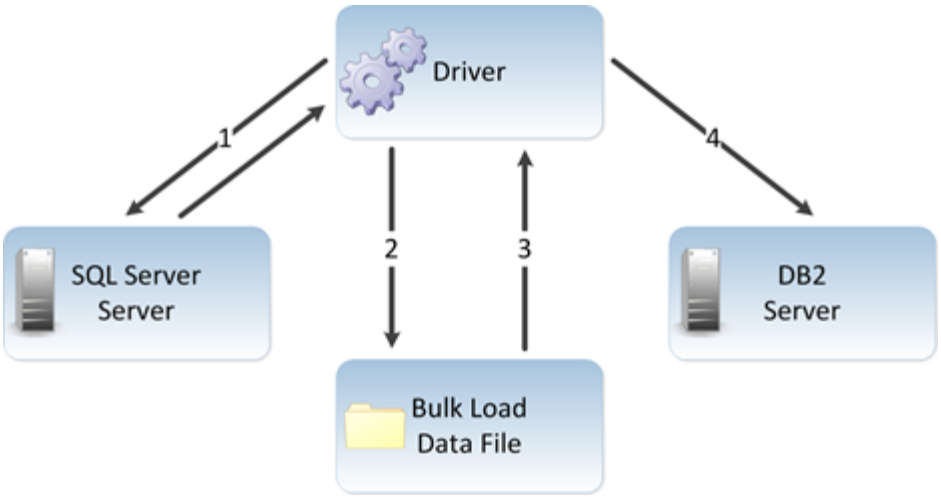
On Windows, UNIX, and Linux, the driver supports DataDirect Bulk Load when connected to databases that are orgMicrosoft SQL Server 2000 and higher. This feature allows your application to send large numbers of rows of data to a database. The driver sends the data to the database in a continuous stream instead of numerous smaller database packets. Similar to batch operations, using bulk load improves performance because far fewer network round trips are required. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations.

DataDirect Bulk Load requires a licensed installation of the drivers. If the drivers are installed with an evaluation license, the bulk load feature is available for prototyping with your applications, but with limited scope. Contact your sales representative or Progress DataDirect SupportLink for further information.

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.

Bulk load operations are accomplished by exporting the results of a query from a database into a comma-separated value (CSV) file, a bulk load data file. The driver then loads the data from bulk load data file into a different database. The file can be used by any DataDirect *for* ODBC driver. In addition, the bulk load data file is supported by other DataDirect product lines that feature bulk loading, for example, a DataDirect Connect for ADO.NET data provider that supports bulk load.

Suppose that you had customer data on an SQL Server server and need to export it to a DB2 server. The driver would perform the following steps:



1. Application using SQL Server Wire Protocol driver sends query to and receives results from SQL Server server.
2. Driver exports results to bulk load data file.
3. Driver retrieves results from bulk load data file.
4. Driver bulk loads results on DB2 server.

Bulk Export and Load Methods

You can take advantage of DataDirect Bulk Load either through the Driver setup dialog or programmatically.

Applications that are already coded to use parameter array batch functionality can leverage DataDirect Bulk Load features through the Enable Bulk Load connection option on the Bulk tab of the Driver setup dialog. Enabling this option automatically converts the parameter array batch operation to use the database bulk load protocol without any code changes to your application.

If you are not using parameter array batch functionality, the bulk operation buttons **Export Table** and **Load Table** on the Bulk tab of the driver Setup dialog also allow you to use bulk load functionality without any code changes. See "Bulk tab" for a description of the Bulk tab.

If you want to integrate bulk load functionality seamlessly into your application, you can include code to use the bulk load functions exposed by the driver.

For your applications to use DataDirect Bulk Load functionality, they must obtain driver connection handles and function pointers, as follows:

1. Use SQLGetInfo with the parameter SQL_DRIVER_HDBC to obtain the driver's connection handle from the Driver Manager.
2. Use SQLGetInfo with the parameter SQL_DRIVER_HLIB to obtain the driver's shared library or DLL handle from the Driver Manager.
3. Obtain function pointers to the bulk load functions using the function name resolution method specific to your operating system. The bulk.c example program shipped with the drivers contains the function resolveName that illustrates how to obtain function pointers to the bulk load functions.

This is detailed in the code samples that follow.

Exporting data from a database

You can export data from a database in one of three ways:

- From a table by using the driver Setup dialog
- From a table by using DataDirect functions
- From a result set by using DataDirect statement attributes

From the DataDirect driver Setup dialog, select the **Bulk** tab and click **Export Table**. See the driver configuration chapter for a description of this procedure.

Your application can export a table using the DataDirect functions ExportTableToFile (ANSI application) or ExportTableToFileW (Unicode application). The application must first obtain driver connection handles and function pointers, as shown in the following example:

```
HDBC      hdbc;
HENV      henv;
void      *driverHandle;
HMODULE      hmod;
PExportTableToFile exportTableToFile;

char      tableName[128];
char      fileName[512];
char      logFile[512];
int       errorTolerance;
int       warningTolerance;
int       codePage;

/* Get the driver's connection handle from the DM.
   This handle must be used when calling directly into the driver. */

rc = SQLGetInfo (hdbc, SQL_DRIVER_HDBC, &driverHandle, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
    EnvClose (henv, hdbc);
    exit (255);
}

/* Get the DM's shared library or DLL handle to the driver. */

rc = SQLGetInfo (hdbc, SQL_DRIVER_HLIB, &hmod, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
    EnvClose (henv, hdbc);
    exit (255);
}
    exportTableToFile = (PExportTableToFile)
    resolveName (hmod, "ExportTableToFile");
if (! exportTableToFile) {
    printf ("Cannot find ExportTableToFile!\n");
    exit (255);
}
```

```
rc = (*exportTableToFile) (
      driverHandle,
      (const SQLCHAR *) tableName,
      (const SQLCHAR *) fileName,
      codePage,
      errorTolerance, warningTolerance,
      (const SQLCHAR *) logFile);
if (rc == SQL_SUCCESS) {
      printf ("Export succeeded.\n");
}
else {
      driverError (driverHandle, hmod);
}
```

Your application can export a result set using the DataDirect statement attributes SQL_BULK_EXPORT and SQL_BULK_EXPORT_PARAMS.

The export operation creates a bulk load data file with a .csv extension in which the exported data is stored. For example, assume that a source table named GBMAXTABLE contains four columns. The resulting bulk load data file GBMAXTABLE.csv containing the results of a query would be similar to the following:

```
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"
```

A bulk load configuration file with and .xml extension is also created when either a table or a result set is exported to a bulk load data file. See "The bulk load configuration file" for an example of a bulk load configuration file.

In addition, a log file of events as well as external overflow files can be created during a bulk export operation. The log file is configured through either the driver Setup dialog Bulk tab, the ExportTableToFile function, or the SQL_BULK_EXPORT statement attribute. The external overflow files are configured through connection options; see "External overflow files" for details.

Bulk loading to a database

You can load data from the bulk load data file into the target database through the DataDirect driver Setup dialog by selecting the Bulk tab and clicking **Load Table**. See "Bulk tab" for a description of this procedure.

Your application can also load data from the bulk load data file into the target database using the using the DataDirect functions LoadTableFromFile (ANSI application) or LoadTableFromFileW (Unicode application). The application must first obtain driver connection handles and function pointers, as shown in the following example:

```
HDBC      hdbc;
HENV      henv;
void      *driverHandle;
HMODULE   hmod;
PLoadTableFromFile loadTableFromFile;
char      tableName[128];
char      fileName[512];
char      configFile[512];
char      logFile[512];
char      discardFile[512];
int       errorTolerance;
int       warningTolerance;
int       loadStart;
int       loadCount;
int       readBufferSize;

/* Get the driver's connection handle from the DM.
   This handle must be used when calling directly into the driver.*/

rc = SQLGetInfo (hdbc, SQL_DRIVER_HDBC, &driverHandle, 0, NULL);
if (rc != SQL_SUCCESS) {
  ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
  EnvClose (henv, hdbc);
  exit (255);
}
/* Get the DM's shared library or DLL handle to the driver. */

rc = SQLGetInfo (hdbc, SQL_DRIVER_HLIB, &hmod, 0, NULL);
if (rc != SQL_SUCCESS) {
  ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
  EnvClose (henv, hdbc);
  exit (255);
}

loadTableFromFile = (PLoadTableFromFile)
  resolveName (hmod, "LoadTableFromFile");
if (! loadTableFromFile) {
  printf ("Cannot find LoadTableFromFile!\n");
  exit (255);
}

rc = (*loadTableFromFile) (
     driverHandle,
     (const SQLCHAR *) tableName,
     (const SQLCHAR *) fileName,
     errorTolerance, warningTolerance,
     (const SQLCHAR *) configFile,
     (const SQLCHAR *) logFile,
     (const SQLCHAR *) discardFile,
     loadStart, loadCount,
     readBufferSize);
if (rc == SQL_SUCCESS) {
     printf ("Load succeeded.\n");
}
```

```
else {
     driverError (driverHandle, hmod);
}
```

Refer to "DataDirect Bulk Load functions" in the *Progress DataDirect for ODBC Drivers Reference* for more information on supported functions.

Use the BulkLoadBatchSize connection attribute to specify the number of rows the driver loads to the data source 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.

A log file of events as well as a discard file that contains rows rejected during the load can be created during a bulk load operation. These files are configured through either the driver Setup dialog Bulk tab or the LoadTableFromFile function.

The discard file is in the same format as the bulk load data file. After fixing reported issues in the discard file, the bulk load can be reissued using the discard file as the bulk load data file.

The bulk load configuration file

A bulk load configuration file is created when either a table or a result set is exported to a bulk load data file. This file has the same name as the bulk load data file, but with an .xml extension.

The bulk load configuration file defines in its metadata the names and data types of the columns in the bulk load data file. The file defines these names and data types based on the table or result set created by the query that exported the data.

It also defines other data properties, such as length for character and binary data types, the character encoding code page for character types, precision and scale for numeric types, and nullability for all types.

When a bulk load data file cannot read its configuration file, the following defaults are assumed:

- All data is read in as character data. Each value between commas is read as character data.
- The default character set is defined, on Windows, by the current Windows code page. On UNIX/Linux, it is the IANAAppCodePage value, which defaults to `4`.

For example, the format of the bulk load data file GBMAXTABLE.csv (discussed in "Exporting data from a database") is defined by the bulk load configuration file, GBMAXTABLE.xml, as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<table codepage="UTF-16LE" 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 is supported by an underlying XML Schema defined at:

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

The bulk load configuration file must conform to the bulk load configuration XML schema. Each bulk export operation generates a bulk load configuration file in UTF-8 format. If the bulk load data file cannot be created or does not comply with the XML Schema described in the bulk load configuration file, an error is generated.

Verification of the bulk load configuration file

You can verify the metadata in the configuration file against the data structure of the target database table. This insures that the data in the bulk load data file is compatible with the target database table structure.

The verification does not check the actual data in the bulk load data file, so it is possible that the load can fail even though the verification succeeds. For example, if you were to update the bulk load data file manually such that it has values that exceed the maximum column length of a character column in the target table, the load would fail.

Not all of the error messages or warnings that are generated by verification necessarily mean that the load will fail. Many of the messages simply notify you about possible incompatibilities between the source and target tables. For example, if the bulk load data file has a column that is defined as an integer and the column in the target table is defined as smallint, the load may still succeed if the values in the source column are small enough that they fit in a smallint column.

To verify the metadata in the bulk load configuration file through the DataDirect driver Setup dialog, select the Bulk tab and click **Verify**. See "Bulk tab" for a description of this procedure.

Your application can also verify the metadata of the bulk load configuration file using the DataDirect functions ValidateTableFromFile (ANSI application) or ValidateTableFromFileW (Unicode application). The application must first obtain driver connection handles and function pointers, as shown in the following example:

```
HDBC      hdbc;
HENV      henv;
void      *driverHandle;
HMODULE   hmod;
PValidateTableFromFile validateTableFromFile;
char      tableName[128];
char      configFile[512];
char      messageList[10240];
SQLLEN    numMessages;
/* Get the driver's connection handle from the DM.
   This handle must be used when calling directly into the driver. */
rc = SQLGetInfo (hdbc, SQL_DRIVER_HDBC, &driverHandle, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
    EnvClose (henv, hdbc);
```

```
    exit (255);
}/* Get the DM's shared library or DLL handle to the driver. */
rc = SQLGetInfo (hdbc, SQL_DRIVER_HLIB, &hmod, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
    EnvClose (henv, hdbc);
    exit (255);
}
validateTableFromFile = (PValidateTableFromFile)
  resolveName (hmod, "ValidateTableFromFile");
if (!validateTableFromFile) {
  printf ("Cannot find ValidateTableFromFile!\n");
  exit (255);
}
messageList[0] = 0;
numMessages = 0;
rc = (*validateTableFromFile) (
      driverHandle,
      (const SQLCHAR *) tableName,
      (const SQLCHAR *) configFile,
      (SQLCHAR *) messageList,
      sizeof (messageList),
      &numMessages);
printf ("%d message%s%s\n", numMessages,
       (numMessages == 0) ? "s" :
       ((numMessages == 1) ? " : " : "s : "),
       (numMessages > 0) ? messageList : "");
if (rc == SQL_SUCCESS) {
       printf ("Validate succeeded.\n");
}else {
    driverError (driverHandle, hmod);
}
```

Sample applications

Progress DataDirect provides a sample application that demonstrates the bulk export, verification, and bulk load operations. This application is located in the \samples\bulk subdirectory of the product installation directory along with a text file named bulk.txt. Please consult bulk.txt for instructions on using the sample bulk load application.

A bulk streaming application is also provided in the \samples\bulkstrm subdirectory along with a text file named bulkstrm.txt. Please consult bulkstrm.txt for instructions on using the bulk streaming application.

Character set conversions

It is most performance-efficient to transfer data between databases that use the same character sets. At times, however, you might need to bulk load data between databases that use different character sets. You can do this by choosing a character set for the bulk load data file that will accommodate all data. If the source table contains character data that uses different character sets, then one of the Unicode character sets, UTF-8, UTF-16BE, or UTF-16LE must be specified for the bulk load data file. A Unicode character set should also be specified in the case of a target table uses a different character set than the source table to

minimize conversion errors. If the source and target tables use the same character set, that set should be specified for the bulk load data file.

A character set is defined by a code page. The code page for the bulk load data file is defined in the configuration file and is specified through either the Code Page option of the Export Table driver Setup dialog or through the IANAAppCodePage parameter of the ExportTableToFile function.

For supported code page values, refer to "Code page values" in the *Progress DataDirect for ODBC Drivers Reference*.

Any character conversion errors are handled based on the value of the Report Codepage Conversion Errors connection option. See the individual driver chapters for a description of this option.

The configuration file may optionally define a second code page value for each character column (`externalfilecodepage`). If character data is stored in an external overflow file (see "External overflow files"), this second code page value is used for the external file.

External overflow files

In addition to the bulk load data file, DataDirect Bulk Load can store bulk data in external overflow files. These overflow files are located in the same directory as the bulk load data file. Different files are used for binary data and character data. Whether or not to use external overflow files is a performance consideration. For example, binary data is stored as hexadecimal-encoded character strings in the main bulk load data file, which increases the size of the file per unit of data stored. External files do not store binary data as hex character strings, and, therefore, require less space. On the other hand, more overhead is required to access external files than to access a single bulk load data file, so each bulk load situation must be considered individually.

The value of the Bulk Binary Threshold connection option determines the threshold, in KB, over which binary data is stored in external files instead of in the bulk load data file. Likewise, the Bulk Character Threshold connection option determines the threshold for character data.

In the case of an external character data file, the character set of the file is governed by the bulk load configuration file. If the bulk load data file is Unicode and the maximum character size of the source data is 1, then the data is stored in its source code page. See "Character set conversions".

The file name of the external file contains the bulk load data file name, a six-digit number, and a ".lob" extension in the following format: *`CSVfilename_nnnnnn`*.lob. Increments start at 000001.lob.

Limitations

- A bulk operation is not allowed in a manual transaction if it is not the first event.
- Once a bulk operation is started, any non-bulk operation is disallowed until the transaction is committed.

Summary of related options for DataDirect Bulk Load

Connection Options: Bulk	Description
Batch Size (BulkLoadBatchSize)	The number of rows that the driver sends to the database at a time during

Connection Options: Bulk	Description
	bulk operations. This value applies to all methods of bulk loading. Default: `1024`
Bulk Binary Threshold (BulkBinaryThreshold)	The maximum size, in KB, of binary data that is exported to the bulk data file. If set to `-1`, all binary data, regardless of size, is written to the bulk data file, not to an external file. If set to `0`, all binary data, regardless of size, is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file. If set to *x*, any binary data exceeding this specified number of KB is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file. **Default:** None
Bulk Character Threshold (BulkCharacterThreshold)	The maximum size, in KB, of character data that is exported to the bulk data file. If set to `-1`, all character data, regardless of size, is written to the bulk data file, not to an external file. If set to `0`, all character data regardless of size, is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file. If set to *x*, any character data exceeding this specified number of KB is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file. Default: `-1`
Bulk Load Threshold (BulkLoadThreshold)	Determines when the driver uses bulk load for insert, update, delete, or batch operations. If the Enable Bulk Load option is set to `1` and the number of rows affected by an insert, update, delete, or batch operation exceeds the threshold specified by this option, the driver uses SQL Server bulk load protocol to perform the operation. If set to `0`, the driver always uses bulk load to execute insert, update, delete, or batch operations. If set to *x*, the driver only uses bulk load if the Enable Bulk Load option is enabled and the number of rows to be updated by an insert, update, delete, or batch operation exceeds the threshold. If the operation times out, the driver returns an error.

Connection Options: Bulk	Description
	Default: 2
Bulk Options (BulkLoadOptions)	Toggles options for the bulk load process. The value specified is a positive integer representing the cumulative total of the Bulk Options values. The following bulk load options are available: • Check Constraints - Checks constraints while data is being inserted. Value=16. • Fire Triggers - Causes the server to fire the insert triggers for rows being inserted into the database. Value=32. • Keep Identity - Preserves source identity values. When not enabled, identity values are assigned by the destination. Value=1. • Keep Nulls - Preserves null values in the destination table regardless of the settings for default values. When not enabled, null values are replaced by column default values, where applicable. Value=64. • Table Lock - Assigns a table lock for the duration of the bulk copy operation. Other applications are not permitted to update the table during the copy operation. When not enabled, the default bulk locking mechanism (row or table) specified by the table lock on bulk load server option is used. Value=2. If disabled, the bulk load operation continues even if a value that would cause an index to be invalidated is loaded. **Note:** The cumulative value of the options is only used in a connection string with the connection string attribute, BulkLoadOptions. On the Bulk tab of the driver Setup dialog, the individual options are enabled by selecting the appropriate check box. **Default:** 2 (Table Lock enabled)
Field Delimiter (BulkLoadFieldDelimiter)	Specifies the character that the driver will use to delimit the field entries in a bulk load data file. **Default:** None
Record Delimiter (BulkLoadRecordDelimiter)	Specifies the character that the driver will use to delimit the record entries in a bulk load data file. **Default:** None

See "Connection option descriptions" for details about configuring the options.

Using IP addresses

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

If your network supports named servers, the server name specified in the data source can resolve to an IPv4 or IPv6 address.

In the following connection string example, the IP address for the SQL Server server is specified in IPv4 format:

```
DRIVER=DataDirect 8.0 SQL Server Wire Protocol;
HostName=123.456.78.90;PORT=1433;
DB=SQLSACCT;UID=JOHN;PWD=XYZZYYou
```

In the following connection string example, the IP address for the SQL Server server is specified in IPv6 format:

```
DRIVER=DataDirect 8.0 SQL Server Wire Protocol;
HostName=2001:DB8:0000:0000:8:800:200C:417A;PORT=1433;
DB=SQLSACCT;UID=JOHN;PWD=XYZZYYou
```

In addition to the normal IPv6 format, the drivers in the preceding tables support IPv6 alternative formats for compressed addresses. For example, the following connection string specifies the server using IPv6 format, but uses the compressed syntax for strings of zero bits:

```
DRIVER=DataDirect 8.0 SQL Server Wire Protocol;
HostName=2001:DB8:0:0:8:800:200C:417A;PORT=1433;
DB=SQLSACCT;UID=JOHN;PWD=XYZZYYou
```

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

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

XA interface support

The driver enables support for distributed transactions by implementing the XA interface. In an X/Open Distributed Transaction Processing (DTP) system, the XA interface provides a method for the Transaction Manager (TM) to call xa_ routines to interact with the Resource Manager (RM).

For more information on the X/Open DTP system, the XA interface, and the XA components discussed in the following procedure, refer to Distributed Transaction Processing: The XA Specification.

To perform distributed transactions:

1. Establish a connection between your application and the RM (the database server you want to connect to) in AUTOCOMMIT_OFF mode. For example:

```
rc = SQLConnect((HDBC)hdbc, (SQLCHAR*)dataSource, SQL_NTS,
(SQLWCHAR*)"user", SQL_NTS, (SQLWCHAR*)"password", SQL_NTS);
rc = SQLSetConnectAttr((HDBC)hdbc, SQL_ATTR_AUTOCOMMIT,
```

```
SQL_AUTOCOMMIT_OFF, SQL_NTS);
```

where:

user

is the user name required to connect to the RM.

password

is the password required to connect to the RM.

2. Load the driver library and call the `GetXaSwitch()` function. `GetXaSwitch()` returns the XA switch data structure (`xa_switch_t`) exposed by the RM. `xa_switch_t` contains function pointers to the `xa_` routines that must be invoked by the TM to interact with the RM. For example:

```
#ifndef WIN32
        LPCSTR libFile = "ddsqls.dll";
        HMODULE sqlsLib = LoadLibrary(libFile);
        xaSwitch = (XaSwitch)GetProcAddress(sqlsLib, "GetXaSwitch");
#else // Unix
        handle = dlopen("ddsqls.so", RTLD_LAZY);
        xaSwitch = (XaSwitch)dlsym(handle, "GetXaSwitch_");
#endif // End of WIN32
        xaSwitch(0, &xaSwitchPtr);
```

3. Establish an XA connection between TM and RM. As a result, a global transaction is created. For example:

```
int RMID = rmid_value;
// Use xa_open_entry with XA connection string, RMID, and flag to establish
the connection.
rc = xaSwitchPtr->xa_open_entry("SSWP_XA+HostName=host_name+
PortNumber=1433+ACC=account_credentials+
SesTM=session_timeout+DataSource=dsn_name",rmid,flag);
```

where:

rmid_value

is the RM identifier.

host_name

is the name or IP address of the RM you want to connect to.

account_credentials

are the credentials required to connect to the RM. They must be provided in the following format: `P/USER/PASSWORD`.

session_timeout

is the number of seconds the session remains active.

dsn_name

is the data source name.

flag

determines the function that is called after the connection ends. For example, `TMASYNC` and `TMNOFLAGS`.

4. Create an XID for the transaction branch you want to associate with the global transaction. For example:

```
void DoMakeXid(int id, XID *xaXid){
int *dataPtr;
memset(xaXid, 0, sizeof(*xaXid));
xaXid->formatID = format_id
xaXid->gtrid_length = gtrid_length;
xaXid->bqual_length = bqual_length;
dataPtr = (int*)&xaXid->data;
dataPtr[0] = GetCurrentThreadId();
dataPtr[1] = GetCurrentThreadId();
}
```

where:

format_id

is the identifier that indicates the naming convention used by the TM.

gtrid_length

is the length of the global transaction ID. It should not exceed 64 bytes.

bqual_length

is the length of the branch qualifier ID. It should not exceed 64 bytes.

Note: In the above example, the GetCurrentThreadId() function returns the global transaction and branch qualifier IDs. However, if you are using a SQL Server TM, you can use the IDs obtained from the TM instead of those returned by the GetCurrentThreadId() function.

---

5. Start the transaction branch, enlist the connection, and then execute the required query.

```
// Use xa_start_entry to start the transaction branch.
rc = xaSwitchPtr->xa_start_entry(&xId, rmid, flag);

// Enlist connection.
rc = SQLSetConnectOption(hDbc, SQL_ATTR_ENLIST_IN_XA, (UDWORD)1);
```

```
// Execute query.
rc = SQLExecDirect(hstmt, (unsigned char*)"INSERT INTO table_name
VALUES(value_1, 'value_2')", SQL_NTS);
```

6. Prepare and commit the transaction branch; then, delist the connection.

```
// Use xa_prepare_entry to prepare the trasaction branch.
rc = xaSwitchPtr->xa_prepare_entry(&xId, rmid, TMNOFLAGS);

// Use xa_commit_entry to commit the transaction branch.
rc = xaSwitchPtr->xa_commit_entry(&xId, rmid, TMNOFLAGS);

// Delist connection.
rc = SQLSetConnectOption(hDbc, SQL_ATTR_ENLIST_IN_XA, (UDWORD)0);
```

7. End the transaction branch.

```
// Use xa_end_entry to end the transaction branch.
rc = xaSwitchPtr->xa_end_entry(&xId, rmid, flag);
```

Binding parameter markers

An ODBC application can prepare a query that contains dynamic parameters. Each parameter in a SQL statement must be associated, or bound, to a variable in the application before the statement is executed. When the application binds a variable to a parameter, it describes that variable and that parameter to the driver. Therefore, the application must supply the following information:

- The data type of the variable that the application maps to the dynamic parameter
- The SQL data type of the dynamic parameter (the data type that the database system assigned to the parameter marker)

The two data types are identified separately using the SQLBindParameter function. You can also use descriptor APIs as described in the Descriptor section of the ODBC specification (version 3.0 and higher).

The driver relies on the binding of parameters to know how to send information to the database system in its native format. If an application furnishes incorrect parameter binding information to the ODBC driver, the results will be unpredictable. For example, the statement might not be executed correctly.

To ensure interoperability, your driver uses only the parameter binding information that is provided by the application.

Isolation and lock levels supported

Microsoft SQL Server supports isolation levels 0 (Read Uncommitted), 1 (Read Committed), 2 (Repeatable Read), and 3 (Serializable). Microsoft SQL Server supports row-level and table-level locking.

Microsoft SQL Server 2005 and higher supports the following additional isolation levels:

- Snapshot
- Read Committed with Snapshots
- Read Committed with Locks (equivalent to Read Committed in previous Microsoft SQL Server versions)

Refer to "Locking and isolation levels" in the *Progress DataDirect for ODBC Drivers Reference* for details.

Using the Snapshot isolation level

The Snapshot isolation level is available only with Microsoft SQL Server 2005 and higher. Setting the SnapshotSerializable connection string attribute 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 minimal or no code changes.

If you are writing a new application, you may want to code it to set the connection attribute SQL_COPT_SS_TXN_ISOLATION to the value SQL_TXN_SS_SNAPSHOT. The application then uses the snapshot isolation level without requiring the Use Snapshot Transactions connection option.

See "Use Snapshot transactions" for additional information.

Number of connections and statements supported

The SQL Server Wire Protocol driver supports multiple connections and multiple statements per connection.

SQL support

The driver supports the core SQL grammar.

Using arrays of parameters

Microsoft SQL Server databases natively support parameter arrays, and the SQL Server Wire Protocol driver, in turn, supports them. When designing an application for performance, using native parameter arrays for bulk inserts or updates, for example, can improve performance.

Refer to "Designing ODBC applications for performance optimization" in the *Progress DataDirect for ODBC Drivers Reference* for details.

Support for Azure Synapse Analytics and Analytics Platform System

The driver transparently connects to Microsoft Azure Synapse Analytics and Microsoft Analytics Platform System (APS); 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 Host Name (HostName) option.
- No support for unquoted identifiers. The driver always enforces ANSI rules regarding quotation marks for all ADW and APS connections (`EnabledQuotedIdentifiers=1`); therefore, the Enable Quoted Identifiers option is disabled.
- 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 Enable Server Side Cursors option is disabled. The driver always disables server side cursors (`EnableServersideCursors=0`).
- No support for XA connections.
- Support for isolation levels is limited to only the read uncommitted level.

  Refer to "Locking and isolation levels" in the *Progress DataDirect for ODBC Drivers Reference* for details.
- Support for the varchar(max), nvarchar(max), varbinary(max) data types is limited to Heap and Clustered Index Tables.
- No support for the following SQL Server data types:

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

  Refer to "String functions" in the *Progress DataDirect for ODBC Drivers Reference* for more information.
- 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	

Refer to "Numeric functions" in the *Progress DataDirect for ODBC Drivers Reference* for more information.

- 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	

Refer to "Date and time functions" in the *Progress DataDirect for ODBC Drivers Reference* for more information.

Inserts on IDENTITY columns in data replication scenarios

The driver supports inserts into IDENTITY columns in data replication scenarios. This functionality may be useful when publishing data from a previous version of SQL Server.

Either the Enable Replication User (EnableReplicationUser) connection option or the `SQL_COPT_REPLICATION_USER` connection attribute (numeric value `1080`) can be used to allow inserts into IDENTITY columns. When the option or attribute is set to `1` (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.

If different values are specified for the Enable Replication User option and the `SQL_COPT_REPLICATION_USER` attribute, driver behavior is determined by the value of the `SQL_COPT_REPLICATION_USER` attribute. The `SQL_COPT_REPLICATION_USER` connection attribute is included in the file qesqlext.h installed with the product.

Packet logging

The driver code includes a packet logging mechanism that allows you to log TCP packets transmitted

between your driver and database over the network layer. The logs compiled from can then be analyzed and used to troubleshoot issues. You can enable and configure logging using driver connection options.

Note: The packet logging mechanism is supported only for drivers that transmit TCP packets. Refer to "Packet Logging" in the *Progress DataDirect for ODBC Drivers Reference* for a list of supported drivers.

See the following "Packet Logging Connection options" section for a list of connection options used to configure packet logging.

To enable TCP packet logging:

1. Configure and enable packet logging using one of the following methods:
   - Driver setup dialog (Windows)
   - odbc.ini file (UNIX/Linux)
   - Connection string

   See the following "Configuring and enabling packet logging" section for details.
2. Start your application and reproduce the issue.
3. Stop the application and disable packet logging.
4. Send your logs to Technical Support for analysis. Optionally, you can view your logs using a text editor.

Configuring and enabling packet logging

The following driver configuration methods can be used to enable and configure packet logging. Note that only the `EnablePacketLogging` connection option is required to enable packet logging. If you do not specify values for the other connection options for packet logging, the default behavior is used.

Driver setup dialog (Windows)

You can specify connection options for packet logging in the Extended Options field of the **Advanced** tab. For example:

```
EnablePacketLogging=1;PacketLoggingFilePrefix=C:\temp\
myPacketLog;PacketLoggingMaxFileSize=7500
```

odbc.ini file (UNIX/Linux)

In your data source definition in the `[ODBC Data Sources]` section of the system information file, you can specify connection options that control packet logging.

```
[SQLServer Wire Protocol]
Driver=ODBCHOME/lib/ivsqls28.so
Description=DataDirect 8.0 SQL Server Wire Protocol
...
Database=SQLSdb1
...
EnablePacketLogging=1
...
LogonID=JOHN
```

```
...
PacketLoggingFilePrefix=/tmp/myPacketLog
...
PacketLoggingMaxFileSize=102400
...
PacketLoggingMaxNumFiles=10
...
Password=secret
...
PortNumber=1433;
...
```

Connection string

You can specify connection options that configure packet logging in connection strings.

```
DRIVER={DataDirect 8.0 SQL Server Wire
Protocol};HostName=SQLServer1;PortNumber=1433;
LogonID=JOHN;Password=secret;Database=SQLSdb1;EnablePacketLogging=1;
PacketLoggingFilePrefix=C:\temp\myPacketLog;
```

Packet logging connection options

The following table describes the connection options used to configure packet logging.

Table 9: Packet Logging Connection Options

Option	Description
`EnablePacketLogging`	If set to `0`, packet logging is disabled. This is the default. If set to `1`, packet logging is enabled. If set to `2`, packet logging is enabled, but the generated log file does not contain packet data. This value is typically used for performance testing. (Windows only) If set to `5`, packet logging and ODBC tracing are enabled. If set to `6`, packet logging and ODBC tracing are enabled, but the log file for packet logging does not contain data.
`PacketLoggingFlush`	If set to `0`, the operating system determines when to write the log content stored in memory to disk. This is the default. If set `1`, the driver determines when to write the log content stored in memory to disk. If set to `2`, the content of memory is written to a the log file after each

Option	Description
	write. This setting provides a more complete logging history in the event of a crash, but can incur a performance penalty.
`PacketLoggingFilePrefix`	Specifies the path and prefix name of the log file. If no path is specified, the trace log resides in the working directory of the application you are using. For example: • /tmp/myLogFile (UNIX/Linux) • C:\temp\myLogFile (Windows) The above examples would generate a file named myLogFile*YYYYMMDDhhmmssxxx_nn*.out in the temp directory. If you do not specify a value for this option, the driver creates log files in the working directory using the following form: pkt*YYYYMMDDhhmmssxxx_nn*.out.
`PacketLoggingMaxFileSize`	Specifies the file size limit (in KB) of the log file. Once this file size limit is reached, a new log file is created and logging continues. The default is `102400`. Note that subsequent files are named by appending sequential numbers, starting at 1, to the end of the original file name, for example, `myLog<timestamp>_1.out`, `myLog<timestamp>_2.out`, and so on.
`PacketLoggingMaxNumFiles`	Specifies the maximum number of log files that can be created. The default is `10`. Once the maximum number of log files is created, the logging mechanism reopens the first file in the sequence, deletes the content, and continues logging in that file until the file size limit is reached, after which it repeats the process with the next file in the sequence.
`PacketLoggingMemBuffSize`	Specifies the maximum amount of memory, in kiloybtes, to use when writing packet logging. The default is `1024`.

5

Connection option descriptions

The following connection option descriptions are listed alphabetically by the GUI name that appears on the driver Setup dialog box. The connection string attribute name, along with its short name, is listed immediately underneath the GUI name.

In most cases, the GUI name and the attribute name are the same; however, some exceptions exist. If you need to look up an option by its connection string attribute name, please refer to the alphabetical table of connection string attribute names.

Also, a few connection string attributes, for example, Password, do not have equivalent options that appear on the GUI. They are in the list of descriptions alphabetically by their attribute names.

Note: The driver does not support specifying values for the same connection option multiple times in a connection string or DSN. If a value is specified using the same attribute multiple times or using both long and short attributes, the connection may fail or the driver may not behave as intended.

The following table lists the connection string attributes supported by the SQL Server Wire Protocol driver.

Table 10: SQL Server Wire Protocol Attribute Names

Attribute (Short Name)	Default
AEKeyCacheTTL (AETTL)	`-1` (No expiration)
AEKeystoreClientSecret (AEKSCS)	None
AEKeyStoreLocation (AEKSL)	None

Attribute (Short Name)	Default
AEKeystorePrincipalId (AEKSPI)	None
AEKeyStoreSecret (AEKSS)	None
AlternateServers (ASRV)	None
AlwaysReportTriggerResults (ARTR)	0 (Disabled)
AnsiNPW (ANPW)	1 (Enabled)
ApplicationIntent (AI)	0 (READWRITE)
ApplicationName (APP)	None
ApplicationUsingThreads (AUT)	1 (Enabled)
AuthenticationMethod (AM)	1 (Encrypt Password)
BulkBinaryThreshold (BBT)	32
BulkCharacterThreshold (BCT)	-1
BulkLoadBatchSize (BLBS)	1024
BulkLoadOptions (BLO)	2
BulkLoadFieldDelimiter (BLFD)	None
BulkLoadRecordDelimiter (BLRD	None
BulkLoadThreshold (BLTH)	2
ColumnEncryption (CE)	Disabled
ConnectionReset (CR)	0 (Disabled)
ConnectionRetryCount (CRC)	0
ConnectionRetryDelay (CRD)	3
CryptoLibName (CLN)	Empty string
CryptoProtocolVersion (CPV)	TLSv1.3, TLSv1.2
Database (DB)	None
DataSourceName (DSN)	None
Description (n/a)	None
Domain (DOM)	None
EnableBulkLoad (EBL)	0 (Disabled)

Attribute (Short Name)	Default
EnableFIPS (EF)	0 (Default provider)
EnableQuotedIdentifiers (EQI)	0 (Disabled)
EnableReplicationUser (ERU)	0 (Disabled)
EnableServersideCursors (ESSC)	1 (Enable All Except Forward Only)
EncryptionMethod (EM)	0 (None)
FailoverGranularity (FG)	0 (Non-Atomic)
FailoverMode (FM)	0 (Connection)
FailoverPreconnect (FP)	0 (Disabled)
FetchTSWTZasTimestamp (FTSWTZAT)	0 (Disabled)
FetchTWFSasTime (FTWFSAT)	1 (Enabled)
GSSClient (GSSC)	native
HostName (HOST)	None
HostNameInCertificate (HNIC)	None
IANAAppCodePage (IACP) (UNIX ONLY)	4 (ISO 8559-1 Latin-1)
InitializationString (IS)	None
KeepAlive (KA)	0 (Disabled)
KeepConnectionActive (KCA)	0 (Disabled)
Language (LANG)	None
LoadBalanceTimeout (LBT)	0
LoadBalancing (LB)	0 (Disabled)
LoginTimeout (LT)	None
LogonID (UID)	None
MaxPoolSize (MXPS)	100
MinPoolSize (MNPS)	0
MultiSubnetFailover (MSF)	0 (Disabled)
OpenSSLConfigFile (OSSLCNF)	*install_dir*\drivers (Windows) *install_dir*/lib (UNIX/Linux)

Attribute (Short Name)	Default
OpenSSLProviderPath (OSSLPP)	*install_dir*\drivers (Windows) *install_dir*/lib (UNIX/Linux)
PacketSize (PS)	-1
Password (PWD)	None
Pooling (POOL)	0 (disabled)
PortNumber (PORT)	1433
PRNGSeedFile (PSF) UNIX\Linux only	/dev/random
PRNGSeedSource (PSS) UNIX\Linux only	0 (File)
ProxyHost (PXHN)	Empty string
ProxyMode (PXM)	0 (NONE)
ProxyPassword (PXPW)	Empty string
ProxyPort (PXPT)	0
ProxyUser (PXU)	Empty string
QueryTimeout (QT)	0
ReportCodepageConversionErrors (RCCE)	0 (Ignore Errors)
SSLLibName (SLN)	Empty string
SnapshotSerializable (SS)	0 (Disabled)
SocketIdleTimeCheckInterval (SITCI)	1500 (seconds)
Truststore (TS)	None
TruststorePassword (TSP)	None
ValidateServerCertificate (VSC)	1 (enabled)
WorkstationID (WSID)	None
XMLDescribeType (XDT)	-10

Related Links

For details, see the following topics:

- AE Keystore Location

- AE Keystore Secret
- Alternate Servers
- Always Report Trigger Results
- AnsiNPW
- Application Intent
- Application Name
- Application Using Threads
- Authentication Method
- Batch Size
- Bulk Binary Threshold
- Bulk Character Threshold
- Bulk Load Threshold
- Bulk Options
- Column Encryption
- Connection Pooling
- Connection Reset
- Connection Retry Count
- Connection Retry Delay
- Crypto Protocol Version
- CryptoLibName
- Data Source Name
- Database
- Description
- Domain
- Enable Bulk Load
- Enable FIPS
- Enable Quoted Identifiers
- Enable Replication User
- Enable Server Side Cursors
- Encryption Method
- Failover Granularity

- Failover Mode
- Failover Preconnect
- Fetch TSWTZ as Timestamp
- Fetch TWFS as Time
- Field Delimiter
- GSS Client Library
- Host Name
- Host Name In Certificate
- IANAAppCodePage
- Initialization String
- Keep Connection Active
- Key Cache Time To Live
- Key Store Principal Id
- Key Store Secret
- Language
- Load Balance Timeout
- Load Balancing
- Login Timeout
- Max Pool Size
- Min Pool Size
- Multi-Subnet Failover
- OpenSSL Config File
- OpenSSL Provider Path
- Packet Size
- Password
- Port Number
- PRNGSeedFile
- PRNGSeedSource
- Proxy Host
- Proxy Mode
- Proxy Password

- Proxy Port
- Proxy User
- Query Timeout
- Record Delimiter
- Report Codepage Conversion Errors
- Socket Idle Time
- SSLLibName
- TCP Keep Alive
- Trust Store
- Trust Store Password
- Use Snapshot Transactions
- User Name
- Validate Server Certificate
- Workstation ID
- XML Describe Type

AE Keystore Location

Attribute

AEKeyStoreLocation (AEKSL)

Purpose

Specifies the absolute path to the PKCS #12 file. This option is used only when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`) and the PKCS #12 file is the keystore provider.

Valid Values

`string`

where:

`string`

is the absolute path to the PKCS #12 file.

Default

None

GUI Tab

Security tab

See Also

- AE Keystore Secret
- Always Encrypted

AE Keystore Secret

Attribute

AEKeyStoreSecret (AEKSS)

Purpose

Specifies the password used to access the PKCS #12 file. This option is used only when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`) and a password-protected PKCS #12 file is the keystore provider.

Valid Values

`string`

where:

`string`

is the password used to access the PKCS #12 file.

Default

None

GUI Tab

Security tab

See Also

- AE Keystore Location
- Always Encrypted

Alternate Servers

Attribute

AlternateServers (ASRV)

Purpose

A list of alternate database servers to which the driver tries to connect if the primary database server is unavailable. Specifying a value for this option enables connection failover for the driver. The value you specify must be in the form of a string that defines the physical location of each alternate server. All of the other required connection information for each alternate server is the same as what is defined for the primary server connection.

Valid Values

```
(HostName=hostvalue:PortNumber=portvalue:Database=databasevalue[, . . .])
```

You must specify the host name, port number, and database name of each alternate server.

Notes

- An alternate server address in IPv6 format must be enclosed in double quotation marks.

Example

The following Alternate Servers value defines two alternate database servers for connection failover:

```
AlternateServers=(HostName=SqlsServer:PortNumber=1433:Database=Sqlsdb1,
HostName=255.201.11.24:PortNumber=1434:Database=Sqlsdb2)
```

Default

None

GUI Tab

Failover tab

Always Report Trigger Results

Attribute

AlwaysReportTriggerResults (ARTR)

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). For Microsoft SQL Server 2005 and higher and Windows Azure SQL Database, this includes triggers that are fired by Data Definition Language (DDL) events.

Valid Values

`0` | `1`

Behavior

If set to `1` (Enabled), the driver returns all results, including results that are generated by triggers. Multiple trigger results are returned one at a time. You can use the SQLMoreResults function to return individual trigger results. Warnings and errors are reported in the results as they are encountered.

If set to `0` (Disabled):

- For Microsoft SQL Server 2005 and higher and Windows Azure SQL Database, the driver does not report trigger results if the statement is a single INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, GRANT, REVOKE, or DENY statement.
- For other Microsoft SQL Server databases, the driver does not report trigger results if the statement is a single INSERT, UPDATE, or DELETE statement.

When set to `0`, the only result that is returned is the update count that is generated by the statement that was executed (if no errors occurred). Although trigger results are ignored, any errors and warnings that are generated by the trigger are reported. If errors are reported, the update count is not reported.

Default

`0` (Disabled)

GUI Tab

Advanced tab

AnsiNPW

Attribute

AnsiNPW (ANPW)

Purpose

Determines whether ANSI-defined behaviors are exposed. Setting this option has no effect on NULL concatenation for Windows Azure SQL Database or SQL Server versions higher than SQL Server 2012.

Valid Values

`0` | `1`

Behavior

When set to `1` (Enabled), the driver sets four ANSI-defined behaviors for handling NULL comparisons: NULLS, character data padding, warnings, and NULL concatenation.

When set to `0` (Disabled), ANSI-defined behaviors are not exposed. If the driver appears to be truncating trailing blank spaces, set this attribute to `0` (Disabled).

Default

`1` (Enabled)

GUI Tab

Advanced tab

Application Intent

Attribute

ApplicationIntent (AI)

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 Always On Availability Groups have been deployed.

Valid Values

`0` | `1`

Behavior

If set to `0` (READWRITE), the driver connects to a read-write node in the Always On environment.

If set to `1` (READONLY), the driver requests read-only routing and connects to the read-only database replicas specified by the server.

Notes

- By setting ApplicationIntent to `1` (ReadOnly) and querying read-only database replicas when possible, you can improve efficiency of your environment by reducing the work load on read-write nodes.
- When ApplicationIntent is enabled, the virtual network name (VNN) of the availability group listener must be specified in the Host Name connection option.

Default

`0` (READWRITE)

GUI Tab

Advanced tab

Application Name

Attribute

ApplicationName (APP)

Purpose

The name the database uses to identify your application.

Valid Values

string

where:

string

is your application name.

Default

None

GUI Tab

Advanced tab

Application Using Threads

Attribute

ApplicationUsingThreads (AUT)

Purpose

Determines whether the driver works with applications using multiple ODBC threads.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), the driver works with single-threaded and multi-threaded applications.

If set to 0 (Disabled), the driver does not work with multi-threaded applications. If using the driver with single-threaded applications, this value avoids additional processing required for ODBC thread-safety standards.

Notes

- This connection option can affect performance.

Default

1 (Enabled)

GUI Tab

Advanced tab

See Also

Performance considerations

Authentication Method

Attribute

AuthenticationMethod (AM)

Purpose

Specifies the method the driver uses to authenticate the user to the server when a connection is established. If the specified authentication method is not supported by the database server, the connection fails and the driver generates an error.

Valid Values

1 | 4 | 9 | 10 | 13 | 36 | 37

Behavior

If set to 1 (Encrypt Password), the driver sends the user ID in clear text and an encrypted password to the server for authentication.

If set to 4 (Kerberos Authentication), the driver uses Kerberos authentication. This method supports both Microsoft Entra Kerberos and MIT Kerberos environments. Setting this value to 4 also enables NTLMv2 and NTLMv1 authentication on Windows platforms. The protocol used for a connection is determined by the local security policy settings for the client.

(UNIX and Linux only) If set to 9 on Linux and UNIX platforms, the driver uses NTLMv1 or NTLMv2 authentication. The driver determines which protocol to use based on the size of the password provided. For passwords 14 bytes or less, the driver uses NTLMv1; otherwise, the driver uses NTLMv2. To connect to the database, users must supply the Windows User Id, Password, and, in some cases, Domain to the driver.

(UNIX and Linux only) If set to 10, the driver uses NTLMv2 authentication. To connect to the database, users

must supply the Windows User Id, Password, and, in some cases, Domain to the driver.

If set to `13` (Active Directory Password), the driver authenticates using an Entra ID user name and password when connecting to a Azure SQL Database data store. All communications between the service are encrypted using TLS/SSL.

Important: Before enabling Entra ID Password authentication, see "Microsoft Entra ID Authentication" for requirements and additional information.

If set to `36` (Active Directory Service Principal), the driver authenticates using an Entra ID service principal when establishing a connection to an Azure SQL Database data store. All communications between the service are encrypted using TLS/SSL.

Important: Before enabling Entra ID Service Principal authentication, see "Microsoft Entra ID Authentication" for requirements and additional information.

If set to `37` (Active Directory Manged Identity), the driver authenticates using a managed identity when accessing Azure resources. All communications between the service are encrypted using TLS/SSL.

Important: Before enabling Entra ID Service Managed Identity, see "Microsoft Entra ID Authentication" for requirements and additional information.

Notes

- NTLM single sign on is supported only on Windows.

Default

`1` (Encrypt Password)

GUI Tab

Security tab

See Also

Microsoft Entra ID authentication

Batch Size

Attribute

BulkLoadBatchSize (BLBS)

Purpose

The number of rows that the driver sends to the database at a time during bulk operations. This value applies to all methods of bulk loading.

Valid Values

`0` | *x*

where:

x

is a positive integer that specifies the number of rows to be sent.

Default

`1024`

GUI Tab

Bulk tab

Bulk Binary Threshold

Attribute

BulkBinaryThreshold (BBT)

Purpose

The maximum size, in KB, of binary data that is exported to the bulk data file.

Valid Values

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

where:

x

is an integer that specifies the number of KB.

Behavior

If set to `-1`, all binary data, regardless of size, is written to the bulk data file, not to an external file.

If set to `0`, all binary data, regardless of size, is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file.

If set to *x*, any binary data exceeding this specified number of KB is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file.

Default

`32`

GUI Tab

Bulk tab

Bulk Character Threshold

Attribute

BulkCharacterThreshold (BCT)

Purpose

The maximum size, in KB, of character data that is exported to the bulk data file.

Valid Values

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

where:

x

is an integer that specifies the number of KB.

Behavior

If set to `-1`, all character data, regardless of size, is written to the bulk data file, not to an external file.

If set to `0`, all character data regardless of size, is written to an external file, not the bulk data file. A reference

to the external file is written to the bulk data file.

If set to *x*, any character data exceeding this specified number of KB is written to an external file, not the bulk data file. A reference to the external file is written to the bulk data file.

Default

-1

GUI Tab

Bulk tab

Bulk Load Threshold

Attribute

BulkLoadThreshold (BLTH)

Purpose

Determines when the driver uses bulk load for insert, update, delete, or batch operations. If the Enable Bulk Load option is set to 1 (enabled) and the number of rows affected by an insert, update, delete, or batch operation exceeds the threshold specified by this option, the driver uses SQL Server bulk load protocol to perform the operation.

Valid Values

0 | *x*

where:

x

is a positive integer that represents a threshold (number of rows).

Behavior

If set to 0, the driver always uses bulk load to execute insert, update, delete, or batch operations.

If set to *x*, the driver only uses bulk load if the Enable Bulk Load option is enabled and the number of rows to be updated by an insert, update, delete, or batch operation exceeds the threshold. If the operation times out, the driver returns an error.

Notes

- If the Enable Bulk Load option is set to `false`, this option is ignored.

Default

`2`

GUI Tab

Bulk tab

Bulk Options

Attribute

BulkLoadOptions (BLO)

Purpose

Toggles options for the bulk load process.

Valid Values

`0` | *x*

where:

x

is a positive integer representing the cumulative total of the Bulk Options values.

Behavior

If set to `0`, none of the options for bulk load are enabled.

If set to *x*, the values represented by x are enabled.

Note: The cumulative value of the options is only used in a connection string with the connection string attribute, BulkLoadOptions. On the Bulk tab of the driver Setup dialog, the individual options are enabled by selecting the appropriate check box.

The following bulk load options are available:

- Check Constraints - Checks constraints while data is being inserted. Value=16.
- Fire Triggers - Causes the server to fire the insert triggers for rows being inserted into the database. Value=32.
- Keep Identity - Preserves source identity values. When not enabled, identity values are assigned by the destination. Value=1.
- Keep Nulls - Preserves null values in the destination table regardless of the settings for default values. When not enabled, null values are replaced by column default values, where applicable. Value=64.
- Table Lock - Assigns a table lock for the duration of the bulk copy operation. Other applications are not permitted to update the table during the copy operation. When not enabled, the default bulk locking mechanism (row or table) specified by the table lock on bulk load server option is used. Value=2.

Example

If you wanted to enable Check Constraints (16), Fire Triggers (32), and Keep Identity (1) in a connection string, you would add the values together:

```
BulkLoadOptions=49
```

To enable these options on the Bulk tab of the driver Setup dialog, you would simply select the check box for each one.

Default

2 (Table Lock enabled)

GUI Tab

Bulk tab

Column Encryption

Attribute

ColumnEncryption (CE)

Purpose

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

The application can override the value for this option by setting a value for the SQL_SOPT_SS_COLUMN_ENCRYPTION statement attribute in an SQL Statement, thereby allowing you to

enable/disable support for the Always Encrypted feature during the same connection. By enabling support only as needed, you can avoid some of the overhead associated with processing queries using Always Encrypted and improve performance. The Column Encryption option must be set to `Enabled` to use the `SQL_SOPT_SS_COLUMN_ENCRYPTION` statement attribute. See the "Enabling Always Encrypted" for more information.

Valid Values

`Disabled` | `Enabled`

Behavior

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`, 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 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.

Notes

- When Always Encrypted functionality is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`), 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.
- If you using the Azure Key Vault as your keystore provider, values for the Key Store Principal ID (AEKeystorePrincipalId) and Key Store Secret (AEKeystoreClientSecret) options must be specified.

Default

`Disabled`

GUI Tab

Security tab

See Also

- Enabling Always Encrypted
- Key Store Principal Id

- Key Store Secret
- Key Cache Time To Live
- Performance considerations
- Always Encrypted

Connection Pooling

Attribute

Pooling (POOL)

Purpose

Specifies whether to use the driver's connection pooling.

Valid Values

`0` | `1`

Behavior

If set to `1` (Enabled), the driver uses connection pooling.

If set to `0` (Disabled), the driver does not use connection pooling.

Notes

- The application must be thread-enabled to use connection pooling.
- This connection option can affect performance.

Default

`0` (Disabled)

GUI Tab

Pooling tab

See Also

Performance considerations

Connection Reset

Attribute

ConnectionReset (CR)

Purpose

Determines whether the state of connections that are removed from the connection pool for reuse by the application is reset to the initial configuration of the connection.

Valid Values

`0` | `1`

Behavior

If set to `1` (Enabled), the state of connections removed from the connection pool for reuse by an application is reset to the initial configuration of the connection. Resetting the state can negatively impact performance because additional commands must be sent over the network to the server to reset the state of the connection.

If set to `0` (Disabled), the state of connections is not reset.

Notes

- This connection option can affect performance.

Default

`0` (Disabled)

GUI Tab

Pooling tab

See Also

Performance considerations

Connection Retry Count

Attribute

ConnectionRetryCount (CRC)

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.

This option and the Connection Retry Delay connection option, which specifies the wait interval between attempts, can be used in conjunction with failover.

Valid Values

`0` | *x*

where:

x

is a positive integer from 1 to 65535.

Behavior

If set to `0`, the driver does not try to connect 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 error that is generated by the last server to which it tried to connect.

Default

`0`

GUI Tab

Failover tab

Connection Retry Delay

Attribute

ConnectionRetryDelay (CRD)

Purpose

Specifies the number of seconds the driver waits between connection retry attempts when Connection Retry Count is set to a positive integer.

This option and the Connection Retry Count connection option can be used in conjunction with failover.

Valid Values

`0` | *x*

where:

x

is a positive integer from 1 to 65535.

Behavior

If set to `0`, there is no delay between retries.

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

Default

`3`

GUI Tab

Failover tab

Crypto Protocol Version

Attribute

CryptoProtocolVersion (CPV)

Purpose

Specifies a comma-separated list of the cryptographic protocols to use when TLS/SSL is enabled using the Encryption Method connection option (`EncryptionMethod=1 | 6 | 7 | 8`). 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 database server, driver behavior is determined by the Encryption Method connection option.

Valid Values

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

where:

cryptographic_protocol

is one of the following cryptographic protocols:

`TLSv1.3 | TLSv1.2`

Example

If your security environment is configured to use TLSv1.3 and TLSv1.2, specify the following values:

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

Notes

- The TLSv1.3 protocol is currently supported only for strict connection encryption (`EncryptionMethod=8`).
- This option is ignored if encryption is disabled (`EncryptionMode=0`).
- Consult your database administrator concerning the data encryption settings of your server.

Default

`TLSv1.3,TLSv1.2`

GUI Tab

Security tab

See also

Encryption Method

CryptoLibName

Attribute

CryptoLibName (CLN)

Purpose

The absolute path for the OpenSSL library file containing the cryptographic library to be used by the data source or connection when TLS/SSL is enabled. The cryptograpic library contains the implementations of cryptographic algorithms the driver uses for data encryption.

This option allows you to designate a different cryptographic library if you encounter issues with the default version or want to use a library that you provide. Common issues that require designating a different library include security vulnerabilities with specific libraries or compatibility issues with your server or application.

Valid Values

*absolute_path**openssl_filename*

where:

absolute_path

is the absolute path to where the OpenSSL file is located

openssl_filename

is the name of the OpenSSL library file containing the cryptographic library to be used by your data source or connection.

Example

C:\Program Files\Progress\DataDirect\ODBC\Drivers\ddopenssl30.dll

Notes

- **Warning**: If you are distributing the driver with your application, you must prevent your end users from setting the value for the CryptoLibName option. The CryptoLibName option provides a method for you to specify a cryptographic library file used for TLS/SSL encryption. However, if exposed, the option can be used to specify files that execute malicious or undesirable code. Refer to "Security best practices for ODBC applications" in the *Progress DataDirect for ODBC Drivers Reference* for more information.
- The value specified for this option should be an absolute path to a mounted drive.
- The OpenSSL library files provided by Progress combine the cryptographic and TLS/SSL libraries into a single file; therefore, when your drivers are using a Progress library file, the values specified for the

CryptoLibName and SSLLibName options should be the same. For non-Progress library files, the libraries may use separate files, which would require unique values to be specified.

- This option can be used to designate OpenSSL libraries not installed by the product; however, the drivers are only certified against libraries provided by Progress.

Default

Empty string

GUI Tab

The value for this option is specified as an option-value pair in the Extended Options field on the Advanced tab. For example:

CryptoLibName=C:\Program Files\Progress\DataDirect\ODBC\drivers\ddopenssl30.dll;

See also

- Advanced tab
- SSLLibName

Data Source Name

Attribute

DataSourceName (DSN)

Purpose

Specifies the name of a data source in your Windows Registry or odbc.ini file.

Valid Values

string

where:

string

is the name of a data source.

Default

None

GUI Tab

General tab

Database

Attribute

Database (DB)

Purpose

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

Valid Values

database_name

where:

database_name

is the name of a valid database.

Default

None

GUI Tab

General tab

Description

Attribute

Description (n/a)

Purpose

Specifies an optional long description of a data source. This description is not used as a runtime connection attribute, but does appear in the ODBC.INI section of the Registry and in the odbc.ini file.

Valid Values

string

where:

string

is a description of a data source.

Default

None

GUI Tab

General tab

Domain

Attribute

UNIX®

Domain (DOM)

Purpose

Specifies the Windows domain that the driver uses when connecting to a SQL Server Instance.

To connect to the database, users must supply the Windows User Id, Password, and, in some cases, domain to the driver. NTLM single sign on is not supported.

Valid Values

string

where:

string

is a valid Windows domain for the user specified by LoginId. This attribute applies only when Authentication Mode is set to 9.

Default

None

GUI Tab

n/a

Enable Bulk Load

Attribute

EnableBulkLoad (EBL)

Purpose

Specifies the bulk load method.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), 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 0 (Disabled), the driver uses standard parameter arrays.

Default

0 (Disabled)

GUI Tab

Bulk tab

Enable FIPS

Attribute

EnableFIPS (EF)

Purpose

Determines whether the OpenSSL library uses cryptographic algorithms from the FIPS provider or the default provider when TLS/SSL encryption is enabled.

Valid Values

`0` | `1`

Behavior

If set to `0`, the OpenSSL library uses cryptographic algorithms from the default provider.

If set to `1`, the OpenSSL library uses cryptographic algorithms from the FIPS provider.

Notes

- The FIPS provider is supported only on the following platforms: Windows 64-bit, Linux 64-bit, and AIX 64-bit. On the other platforms, the driver uses the default provider of the OpenSSL 3.0 library.
- Do not set the Truststore Password (TruststorePassword) connection option when using the FIPS provider. The truststore password uses the PKCS12KDF algorithm, which is not an approved FIPS algorithm. Hence, it must not be specified when using the FIPS provider.
- For using the FIPS and default providers, the certificates must be encrypted with the OpenSSL 3.0-compliant cryptographic algorithms. See "Generating TLS/SSL certificates with OpenSSL 3.0-compliant algorithms" for more information.

Default

`0`

Enable Quoted Identifiers

Attribute

EnableQuotedIdentifiers (EQI)

Purpose

Determines whether the driver allows the use of quoted identifiers.

Valid Values

`0` | `1`

Behavior

If set to `1` (Enabled), the database enforces ANSI rules regarding quotation marks. Double quotation marks can only be used for identifiers, such as column and table names. Character strings must be enclosed in single quotation marks, for example:

```
SELECT "au_id"
FROM "authors"
WHERE "au_lname" = 'O''Brien'
```

If set to `0` (Disabled), applications that use quoted identifiers encounter errors when they generate SQL statements with quoted identifiers.

Default

`0` (Disabled)

GUI Tab

Advanced tab

Enable Replication User

Attribute

EnableReplicationUser (ERU)

Purpose

Specifies whether explicit values may be inserted into IDENTITY columns defined as NOT FOR REPLICATION.

Valid Values

`0` | `1`

Behavior

If set to `1` (Enabled), the driver allows explicit inserts on IDENTITY columns defined as NOT FOR REPLICATION.

If set to `0` (Disabled), the driver enforces constraints on IDENTITY columns imposed by the NOT FOR REPLICATION flag.

Notes

- To provide greater control to an application, enable replication user functionality may be enabled by setting the `SQL_COPT_REPLICATION_USER` connection attribute (numeric value `1080`) to `1` (Enabled). If different values are specified for the Enable Replication User option and the `SQL_COPT_REPLICATION_USER` attribute, driver behavior is determined by the value of the `SQL_COPT_REPLICATION_USER` attribute.

Default

`0` (Disabled)

GUI Tab

Advanced tab

See also

- Inserts on IDENTITY columns in data replication scenarios

Enable Server Side Cursors

Attribute

EnableServersideCursors (ESSC)

Purpose

Determines whether server-side cursors are enabled for the data source. This option applies to Forward Only, Static and Keyset cursors.

Valid Values

`0` | `1` | `2` | `3` | `4`

Behavior

If set to `0` (Disabled), all server-side cursors are disabled for the data source.

If set to `1` (Enable All Except Forward Only), all server-side scrollable cursors are enabled for the data source, while forward-only cursors on the server side are disabled.

If set to `2` (Enable Forward Only for Rowset Size >1), only forward-only cursors on the server-side are enabled when the rowset size is set to a value greater than one.

If set to `3` (Enable All), all server-side cursors, scrollable and forward-only, are enabled for the data source.

If set to `4` (Enable Forward Only for Select For Update), forward-only cursors on the server-side are enabled only for Select For Update statements. For other Select statements, the driver uses forward-only cursors on the client-side. This setting avoids using driver emulation for other Select statements, thereby improving performance and allowing the use of native updatable result sets.

Notes

- This connection option can affect performance.

Default

`1` (Enable All Except Forward Only)

GUI Tab

Advanced tab

See also

Performance considerations

Encryption Method

Attribute

EncryptionMethod (EM)

Purpose

The method the driver uses to encrypt data sent between the driver and the database server.

Valid Values

`0` | `1` | `6` | `7` | `8`

Behavior

If set to `0` (None), data is not encrypted.

If set to `1` (SSL), data is encrypted using the TLS/SSL protocols specified in the Crypto Protocol Version (CryptoProtocolVersion) connection option.

If set to `6` (RequestSSL), the login request and data are encrypted using TLS/SSL if the server is configured for TLS/SSL. If the server is not configured for TLS/SSL, an unencrypted connection is established. The TLS/SSL protocol used is determined by the setting of the Crypto Protocol Version (CryptoProtocolVersion) connection option.

If set to `7` (LoginSSL), the login request is encrypted using TLS/SSL regardless of whether the server is configured for TLS/SSL. The data is encrypted using TLS/SSL if the server is configured for TLS/SSL, and the data is unencrypted if the server is not configured for TLS/SSL. The TLS/SSL protocol used is determined by the setting of the Crypto Protocol Version (CryptoProtocolVersion) connection option.

If set to `8` (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=1) for the connection, regardless of the setting of the Validate Server Certificate option.
- You must specify a truststore containing the server certificate against which the server will be validated at connection.

Notes

- 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=1`).
- For values `1` through `8`, the TLS/SSL protocol used is determined by the setting of the Crypto Protocol Version (CrytpoProtocolVersion) connection option.
- The driver must use the server-specified packet size when using TLS/SSL encryption. If TLS/SSL is used, any value set for the Packet Size connection option is ignored.
- This connection option can affect performance.
- When using FIPS and default providers, the certificates must be generated using the OpenSSL 3.0-compliant cryptographic algorithms.

Default

`0` (None)

GUI Tab

Security tab

See Also

Crypto Protocol Version

Performance considerations

Failover Granularity

Attribute

FailoverGranularity (FG)

Purpose

Determines whether the driver fails the entire failover process or continues with the process if errors occur while trying to reestablish a lost connection.

This option applies only when Failover Mode is set to `1` (Extended Connection) or `2` (Select).

The Alternate Servers option specifies one or multiple alternate servers for failover and is required for all failover methods.

Valid Values

0 | 1 | 2 | 3

Behavior

If set to 0 (Non-Atomic), the driver continues with the failover process and posts any errors on the statement on which they occur.

If set to 1 (Atomic) the driver fails the entire failover process if an error is generated as the result of anything other than executing and repositioning a Select statement. If an error is generated as a result of repositioning a result set to the last row position, the driver continues with the failover process, but generates a warning that the Select statement must be reissued.

If set to 2 (Atomic Including Repositioning), the driver fails the entire failover process if any error is generated as the result of restoring the state of the connection or the state of work in progress.

If set to 3 (Disable Integrity Check), the driver does not verify that the rows that were restored during the failover process match the original rows. This value applies only when Failover Mode is set to 2 (Select).

Default

0 (Non-Atomic)

GUI Tab

Failover tab

Failover Mode

Attribute

FailoverMode (FM)

Purpose

Specifies the type of failover method the driver uses.

The Alternate Servers option specifies one or multiple alternate servers for failover and is required for all failover methods.

Valid Values

0 | 1 | 2

Behavior

If set to `0` (Connection), the driver provides failover protection for new connections only.

If set to `1` (Extended Connection), the driver provides failover protection for new and lost connections, but not any work in progress.

If set to `2` (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.

Notes

- This connection option can affect performance.

Default

`0` (Connection)

GUI Tab

Failover tab

See Also

Performance considerations

Failover Preconnect

Attribute

FailoverPreconnect (FP)

Purpose

Specifies whether the driver tries to connect to the primary and an alternate server at the same time.

This attribute applies only when Failover Mode is set to `1` (Extended Connection) or `2` (Select) and at least one alternate server is specified.

The Alternate Servers option specifies one or multiple alternate servers for failover and is required for all failover methods.

Valid Values

0 | 1

Behavior

If set to 0 (Disabled), 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.

If set to 1 (Enabled), 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.

Default

0 (Disabled)

GUI Tab

Failover tab

Fetch TSWTZ as Timestamp

Attribute

FetchTSWTZasTimestamp (FTSWTZAT)

Purpose

Determines whether the driver returns column values with the timestamp with time zone data type as the ODBC data type SQL_TYPE_TIMESTAMP or SQL_VARCHAR.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), the driver returns column values with the timestamp with time zone data type as the ODBC type SQL_TYPE_TIMESTAMP. The time zone information in the fetched value is truncated. Use this value if your application needs to process values the same way as TIMESTAMP columns.

If set to 0 (Disabled), the driver returns column values with the timestamp with time zone data type as the ODBC data type SQL_VARCHAR. Use this value if your application requires the time zone information in the fetched value.

Default

0 (Disabled)

GUI Tab

Advanced tab

Fetch TWFS as Time

Attribute

FetchTWFSasTime (FTWFSAT)

Purpose

Determines whether the driver returns column values with the time data type as the ODBC data type SQL_TYPE_TIME or SQL_TYPE_TIMESTAMP.

Supported only for Microsoft SQL Server 2008.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), the driver returns column values with the time data type as the ODBC data type SQL_TYPE_TIME. The fractional seconds portion of the value is truncated.

If set to 0 (Disabled), the driver returns column values with the time data type as the ODBC data type SQL_TYPE_TIMESTAMP. The fractional seconds portion of the value is preserved. Time columns are not searchable when they are described and fetched as timestamp.

Notes

- When returning time with fractional seconds data as SQL_TYPE_TIMESTAMP, the Year, Month and Day parts of the timestamp must be set to zero.

Default

1 (Enabled)

GUI Tab

Advanced tab

Field Delimiter

Attribute

BulkLoadFieldDelimiter (BLFD)

Purpose

Specifies the character that the driver will use to delimit the field entries in a bulk load data file.

Valid Values

x

where:

x

is any printable character.

For simplicity, avoid using a value that can be in the data, including all alphanumeric characters, the dash(-), the colon(:), the period (.), the forward slash (/), the space character, the single quote (') and the double quote ("). You can use some of these characters as delimiters if all of the data in the file is contained within double quotes.

Notes

- The Bulk Load Field Delimiter character must be different from the Bulk Load Record Delimiter.

Default

None

GUI Tab

Bulk tab

GSS Client Library

Attribute

GSSClient (GSSC)

Purpose

The name of the GSS client library that the driver uses to communicate with the Key Distribution Center (KDC).

The driver uses the path defined by the PATH environment variable for loading the specified client library.

Valid Values

`native` | *`client_library`*

where:

`client_library`

is a GSS client library installed on the client.

Behavior

If set to *`client_library`*, the driver uses the specified GSS client library.

If set to native, the driver uses the GSS client shipped with the operating system.

Notes

- **Warning**: If you are distributing the driver with your application, you must prevent your end users from setting the value for the GSS Client Library option. The GSS Client Library option provides a method for you to specify a library file used to communicate with the Key Distribution Center (KDC). However, if exposed, the option can be used to specify files that execute malicious or undesirable code. Refer to "Security best practices for ODBC applications" in the *Progress DataDirect for ODBC Drivers Reference* for more information.
- The value specified for this option should be an absolute path to a mounted drive.

Default

native

GUI Tab

Security tab

Host Name

Attribute

HostName (HOST)

Purpose

The name, IP address, or alias of the server or instance to which you want to connect.

Valid Values

alias_name | *IP_address* | *named_server* | *named_instance* | *server_name* | *virtual_network_name*

where:

alias_name

is the alternate name of the server to which you want to connect. When specifying an alias name, the driver uses the server name and port number value provided by the alias for the connection. Aliases are created using the SQL Server Configuration Manager. Note that aliases are supported only on Windows platforms.

IP_address

is the IP address of the server to which you want to connect. Specify this address as: *IP_address*. For example, you can enter 199.226.224.34.

The IP address can be specified in either IPv4 or IPv6 format. See "Using IP addresses" for details about these formats.

named_server

is the named server address of the server to which you want to connect. Specify this address as: *named_server*. For example, you can enter SSserver.

named_instance

is a named instance of Microsoft SQL Server, Windows Azure SQL Database, Microsoft Azure Synapse Analytics, or Microsoft Analytics Platform System. Specify this address as: *server_name**instance_name*.

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.
- When an alias name is specified with this option, port numbers settings provided by the alias take precedence over the value of the Port Number (PortNumber) connection option.
- If only a server name is specified with no instance name, the driver uses the default instance on the server.
- If only a server name is specified with a backward slash `\` or `\*` at the end with no instance name, the driver uses the first instance on the server with a TCP port.

Default

None

GUI Tab

General tab

Host Name In Certificate

Attribute

HostNameInCertificate (HNIC)

Purpose

A host name for certificate validation when TLS/SSL encryption is enabled and validation is enabled (`Validate Server Certificate=1`). This option provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

Valid Values

host_name | #SERVERNAME#

where:

host_name

is the host name specified in the certificate. Consult your TLS/SSL administrator for the correct value.

Behavior

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

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

Default

None

GUI Tab

Security tab

IANAAppCodePage

Attribute

UNIX® IANAAppCodePage (IACP)

Purpose

An Internet Assigned Numbers Authority (IANA) value. You must specify a value for this option if your application is not Unicode-enabled or if your database character set is not Unicode.

The driver uses the specified IANA code page to convert "W" (wide) functions to ANSI.

The driver and Driver Manager both check for the value of IANAAppCodePage in the following order:

- In the connection string
- In the Data Source section of the system information file (odbc.ini)
- In the ODBC section of the system information file (odbc.ini)

If the driver does not find an IANAAppCodePage value, the driver uses the default value of 4 (ISO 8859-1 Latin-1).

Valid Values

IANA_code_page

where:

IANA_code_page

is one of the valid values listed in "IANAAppCodePage values" in the *Progress DataDirect for ODBC Drivers Reference*. The value must match the database character encoding and the system locale.

Default

4 (ISO 8559-1 Latin-1)

GUI Tab

Advanced tab

See Also

Refer to "Internationalization, localization, and Unicode" in the *Progress DataDirect for ODBC Drivers Reference* for details.

Initialization String

Attribute

InitializationString (IS)

Purpose

A SQL command that is issued immediately after connecting to the database to manage session settings.

Valid Values

SQL_command

where:

SQL_command

is a valid SQL command that is supported by the database.

Example

To set the date format on every connection, specify:

```
Set DateStyle='ISO, MDY'
```

Notes

- If the statement fails to execute, the connection fails and the driver reports the error returned from the server.

Default

None

GUI Tab

Advanced tab

Keep Connection Active

Attribute

KeepConnectionActive (KCA)

Purpose

Specifies whether the driver periodically sends lightweight SQL operations to the database after a connection has been idle for the time specified by the Socket Idle Time (`SocketIdleTimeCheckInterval`) option. The SQL operation resets the Azure SQL Gateway or database idle timeout timer to keep the connection open.

Valid Values

`0` | `1`

Behavior

If set to `0` (Disabled), the driver does *not* send lightweight SQL operations the database to keep the connection open. Once a connection is idle for the duration specified by the Azure SQL Gateway or database, the connection times out.

If set to `1` (Enabled), the driver periodically sends lightweight SQL operations to the database to keep the connection active. Once a connection is idle for the duration specified by the Socket Idle Time option, the driver executes a lightweight query (`Select 0`) to the database to prevent the connection from timing out.

Default

`0` (Disabled)

Notes

- This option differs from TCP Keep Alive (KeepAlive) in that it maintains the connection to the database or Azure SQL Gateway, while TCP Keep Alive maintains only the TCP socket. If you have a database or gateway timeout, you will need to enable this functionality to prevent your connection from terminating during idle periods.

GUI Tab

Advanced tab

See Also

- Socket Idle Time

Key Cache Time To Live

Attribute

AEKeyCacheTTL (AETTL)

Purpose

Determines whether the driver caches column encryption keys. This option is used when Always Encrypted

is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`).

Valid Values

`-1` | `0`

Behavior

If set to `-1`, the driver caches column encryption keys on a per connection basis. The keys remain in the cache until the connection is closed or the application exits.

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

Notes

- Column encryption keys do not persist beyond the life of a 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 eliminating the overhead associated with fetching and decrypting the keys for the same data multiple times during a connection.
- While caching column encryption keys can improve performance, they are designed to be deleted from the cache as a security measure. Therefore, we do not recommend caching keys for applications that remain connected for long periods of time.

Default

`-1` (No expiration)

GUI Tab

Security tab

See Also

- Column Encryption
- Always Encrypted
- Performance considerations

Key Store Principal Id

Attribute

AEKeystorePrincipalId (AEKSPI)

Purpose

Specifies the principal ID used to authenticate against Azure Key Vault. This option 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

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 Key Store Secret (AEKeystoreClientSecret) connection option.
- The driver currently supports only Azure App Registration as the principal ID.
- This option is used only when the Azure Key Vault is specified as the keystore provider in the encryption metadata for result set columns or in statement parameters.
- The driver determines which keystore provider to use based on the encryption metadata received from the server.

Default

None

GUI Tab

Security tab

See Also

- Column Encryption
- Key Store Secret
- Always Encrypted

Key Store Secret

Attribute

AEKeystoreClientSecret (AEKSCS)

Purpose

Specifies the Client Secret used to authenticate against the Azure Key Vault. This option 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 Key Store Principal Id (AEKeystorePrincipalId) connection option.
- This option is used only when the Azure Key Vault is specified as the keystore provider in the encryption metadata for result set columns or in statement parameters.
- The driver determines which keystore provider to use based on the encryption metadata received from the server.

Default

None

GUI Tab

Security tab

See Also

- Column Encryption

- Key Store Principal Id
- Always Encrypted

Language

Attribute

Language (LANG)

Purpose

The national language to use for Microsoft SQL Server system messages.

Valid Values

lang

where:

lang

is the language to use for Microsoft SQL Server system messages. This overrides the default language specified for the login on the server. If no language is specified, the connection uses the default language specified for the login on the server.

Default

None

GUI Tab

Advanced tab

Load Balance Timeout

Attribute

LoadBalanceTimeout (LBT)

Purpose

The number of seconds to keep inactive connections open in a connection pool. An inactive connection is a database session that is not associated with an ODBC connection handle, that is, a connection in the pool that is not in use by an application.

Valid Values

`0 | x`

where:

x

is a positive integer that specifies a number of seconds.

Behavior

If set to `0`, inactive connections are kept open.

If set to *x*, inactive connections are closed after the specified number of seconds passes.

Notes

- The Min Pool Size option may cause some connections to ignore this value.
- This connection option can affect performance.

Default

`0`

GUI Tab

Pooling tab

See also

Performance considerations

Load Balancing

Attribute

LoadBalancing (LB)

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 Alternate Servers option.

Valid Values

`0` | `1`

Behavior

If set to `1` (Enabled), the driver uses client load balancing and attempts to connect to the database servers (primary and alternate servers) in random order.

If set to `0` (Disabled), 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).

Notes

- This option has no effect unless alternate servers are defined for the Alternate Servers connection option.

Default

`0` (Disabled)

GUI Tab

Failover tab

Login Timeout

Attribute

LoginTimeout (LT)

Purpose

The number of seconds the driver waits for a connection to be established before returning control to the application and generating a timeout error. To override the value that is set by this connection option for an individual connection, set a different value in the SQL_ATTR_LOGIN_TIMEOUT connection attribute using the SQLSetConnectAttr() function.

Valid Values

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

where:

x

is a positive integer that represents a number of seconds.

Behavior

If set to `-1`, the connection request does not time out. The driver silently ignores the SQL_ATTR_LOGIN_TIMEOUT attribute.

If set to `0`, the connection request does not time out, but the driver responds to the SQL_ATTR_LOGIN_TIMEOUT attribute.

If set to *x*, the connection request times out after the specified number of seconds unless the application overrides this setting with the SQL_ATTR_LOGIN_TIMEOUT attribute.

Default

`15`

GUI Tab

Advanced tab

Max Pool Size

Attribute

MaxPoolSize (MXPS)

Purpose

The maximum number of connections allowed within a single connection pool. When the maximum number of connections is reached, no additional connections can be created in the connection pool.

Valid Values

An integer from `1` to `65535`

For example, if set to `20`, the maximum number of connections allowed in the pool is `20`.

Notes

- This connection option can affect performance.

Default

`100`

GUI Tab

Pooling tab

See also

Performance considerations

Min Pool Size

Attribute

MinPoolSize (MNPS)

Purpose

The minimum number of connections that are opened and placed in a connection pool, in addition to the active connection, when the pool is created. The connection pool retains this number of connections, even when some connections exceed their Load Balance Timeout value.

Valid Values

`0` | *x*

Behavior

If set to `0`, no connections are opened in addition to the current existing connection.

If set to *x*, the start-up number of connections in the pool is *x* in addition to the current existing connection.

Notes

- This connection option can affect performance.

Example

If set to `5`, the start-up number of connections in the pool is `5` in addition to the current existing connection.

Default

`0`

GUI Tab

Pooling tab

See also

Performance considerations

Multi-Subnet Failover

Attribute

MultiSubnetFailover (MSF)

Purpose

Determines whether the driver attempts parallel connections to the failover IP addresses of an Availability Group during a multi-subnet failover. When Multi-Subnet Failover is enabled, the driver simultaneously attempts to connect to all IP addresses associated with the Availability Group listener when the 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

`1` | `0`

Behavior

If set to `1` (Enabled), the driver attempts parallel connections to all failover IP addresses in an Availability Group when the connection is broken or the listener IP address is 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 `0` (Disabled), the driver uses the failover method specified by the Failover Mode connection option 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 by the Host Name connection option.
- When MultiSubnetFailover is enabled, the Alternate Servers, Load Balancing, and Failover Preconnect connection options are disabled.

Default

`0` (Disabled)

GUI Tab

Failover tab

OpenSSL Config File

Attribute

OpenSSLConfigFile (OSSLCNF)

Purpose

Specifies the absolute path to the configuration file required to load the FIPS provider when the driver is

configured to use OpenSSL 3.0 with FIPS provider for TLS/SSL encryption (`EnableFIPS=1`).

Valid Values

fips_config_file

where:

fips_config_file

is the absolute path to the configuration file. For example: /opt/Progress/DataDirect/ODBC/lib/openssl.cnf.

Notes

- The OpenSSL Config File option is not available on the setup dialog box. To set a value for it, use the Extended Options connection option, which is available on the Advanced tab of the setup dialog box.

Default

- *install_dir*\drivers\openssl.cnf (Windows)
- *install_dir*/lib/openssl.cnf (UNIX/Linux)

OpenSSL Provider Path

Attribute

OpenSSLProviderPath (OSSLPP)

Purpose

Specifies the path to the directory that contains the provider library when TLS/SSL encryption is enabled.

Valid Values

provider_path

where:

provider_path

is the path to the directory that contains the provider library.

Notes

- The OpenSSL Provider Path option is not available on the setup dialog box. To set a value for it, use the Extended Options connection option, which is available on the Advanced tab of the setup dialog box.

Default

- *install_dir*\drivers (Windows)
- *install_dir*/lib (UNIX/Linux)

Packet Size

Attribute

PacketSize (PS)

Purpose

Determines the number of bytes for each database protocol packet that is transferred from the database server to the client machine. 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*

Behavior

If set to `-1`, the driver uses the maximum packet size that is set by the database server.

If set to `0`, the driver uses the default packet size that is used by the database server.

If set to *x*, an integer from `1` to `127`, the driver uses a packet size that is a multiple of 512 bytes. For example, `PacketSize=8` means to set the packet size to 8 * 512 bytes (4096 bytes).

Notes

- If TLS/SSL encryption is used, the driver must use the packet size that is specified by the server. Any value set for this option or the SQL_PACKET_SIZE connect option is ignored if TLS/SSL encryption is used.

- The ODBC connection option SQL_PACKET_SIZE provides the same functionality as the Packet Size option; however SQL_PACKET SIZE and the Packet Size option are mutually exclusive. If Packet Size is specified, the driver returns the message Driver Not Capable if an application attempts to call SQL_PACKET_SIZE. If you do not set the Packet Size option, application calls to SQL_PACKET_SIZE are accepted by the driver.

Default

`-1`

GUI Tab

Advanced tab

See also

Performance considerations

Password

Attribute

Password (PWD)

Purpose

Specifies the password that the application uses to connect to your database or, when using Entra ID Service Principal authentication (`AuthenticationMethod=36`), the client secret for your Azure application.

The Password option cannot be specified through the driver Setup dialog box and should not be stored in a data source. It is specified through the Logon dialog box or a connection string.

Valid Values

`password`|*`client_secret`*

where:

`password`

is a valid password.

client_secret

is a the client secret for your Entra ID application when using Entra ID Service Principal authentication. The value of this property is the same as the ClientSecret in the Entra ID interface.

Default

None

GUI Tab

n/a

Port Number

Attribute

PortNumber (PORT)

Purpose

The port number of the server listener.

Valid Values

port_name

where:

port_name

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

Default

1433

GUI Tab

General tab

PRNGSeedFile

Attribute

PRNGSeedFile (PSF)

Purpose

UNIX®Specifies the absolute path for the entropy-source file or device used as a seed for TLS/SSL key generation.

Valid Values

string | RANDFILE

where:

string

is the absolute path for the entropy-source file or device that seeds the random number generator used for TLS/SSL key generation.

Behavior

If set to *string*, the specified entropy-source file or device seeds the random number generator used for TLS/SSL key generation. Entropy levels and behavior may vary for different files and devices. See the following section for a list of commonly used entropy sources and their behavior.

If set to RANDFILE, the RAND_file_name() function in your application generates a default path for the random seed file. The seed file is $RANDFILE if that environment variable is set; otherwise, it is $HOME/.rnd. If $HOME is not set either, an error occurs.

Common Valid Values

Although other entropy-source files may be specified, the following valid values are for files and devices that are commonly used for seeding:

/dev/random

is a pseudorandom number generator (blocking) that creates a seed from random bits of environmental noise it collects in an entropy pool. When there is insufficient noise in the pool, the file blocks calls until enough noise is collected. This provides more secure TLS/SSL key generation, but at the expense of blocked calls.

/dev/urandom

is a pseudorandom number generator (non-blocking) that creates seeds from random bits from environmental noise it collects in an entropy pool. When there is insufficient noise in the pool, the file reuses bits from the pool instead of blocking calls. This eliminates potential delays associated with blocked calls, but may result in less secure TLS/SSL key generation.

/dev/hwrng

is a hardware random number generator. The behavior is dependent on the device used in your environment.

Notes

- **Warning**: If you are distributing the driver with your application, you must prevent your end users from setting the value for the PRNGSeedFile option. The PRNGSeedFile option provides a method for you to specify a entropy-source file used for TLS/SSL encryption. However, if exposed, the option can be used to specify files that execute malicious or undesirable code. Refer to "Security best practices for ODBC applications" in the *Progress DataDirect for ODBC Drivers Reference* for more information.
- The value specified for this option should be an absolute path to a mounted drive.
- This option is ignored when TLS/SSL is disabled (`EncryptionMethod=0`) or the seed source is set to Poll Only (`PRNGSeedSource=1`).
- For processes that employ multiple TLS/SSL-enabled drivers, the behavior of this option for all drivers is determined by the values specified for the driver that first connects to the process and loads the OpenSSL library. Since the OpenSSL library loads only once per process, the values specified for drivers that subsequently connect are ignored. To ensure that the correct security settings are used, we recommend configuring this option identically for all drivers used in a process.

Default

/dev/random

GUI tab

NA

See also

PRNGSeedSource

PRNGSeedSource

Attribute

PRNGSeedSource (PSS)

Purpose

UNIX®Specifies the source of the seed the driver uses for TLS/SSL key generation. Seeds are a pseudorandom or random value used to set the initial state of the random number generator used to generate TLS/SSL keys. Using seeds with a higher level of entropy, or randomness, provides a more secure transmission of data encrypted using TLS/SSL.

Valid Values

`0` | `1`

Behavior

If set to `0` (File), the driver uses entropy-source file or device specified in the PRNGSeedFile connection option as the seed used for TLS/SSL key generation.

If set to `1` (Poll Only) , the driver uses the RAND_poll function in TLS/SSL to create the seed used for TLS/SSL key generation.

Notes

- For processes that employ multiple TLS/SSL-enabled drivers, the behavior of this option for all drivers is determined by the values specified for the driver that first connects to the process and loads the OpenSSL library. Since the OpenSSL library loads only once per process, the values specified for drivers that subsequently connect are ignored. To ensure that the correct security settings are used, we recommend configuring this option identically for all drivers used in a process.
- This option is ignored when TLS/SSL is disabled (`EncryptionMethod=0`)

Default

`0` (File)

GUI Tab

NA

See also

PRNGSeedFile

Proxy Host

Attribute

ProxyHost (PXHN)

Purpose

Specifies the Hostname and possibly the Domain of the Proxy Server. The value specified can be a host name, a fully qualified domain name, or an IPv4 or IPv6 address.

Valid Values

server_name | *IP_address*

where:

server_name

is the name of the server or a fully qualified domain name to which you want to connect.

The IP address can be specified in either IPv4 or IPv6 format, or a combination of the two. See "Using IP addresses" for details about these formats.

Default

Empty string

Notes

- When proxy mode is disabled (`ProxyMode=0`), the Proxy Host option is ignored.

GUI Tab

General tab

See Also

- Using IP addresses
- Proxy Mode
- Proxy Password
- Proxy Port
- Proxy User

Proxy Mode

Attribute

ProxyMode (PXM)

Purpose

Determines whether the driver connects to your data source endpoint through an HTTP proxy server.

Valid Values

`0` | `1`

Behavior

If set to `0` (NONE), the driver connects directly to the data source endpoint specified by the Host Name connection option.

If set to `1` (HTTP), the driver connects to the data source endpoint through the HTTP proxy server specified by the ProxyHost connection option.

Default

`0` (NONE)

GUI Tab

General tab

See Also

- Proxy Host

- Host Name
- Proxy Password
- Proxy Port
- Proxy User

Proxy Password

Attribute

ProxyPassword (PXPW)

Purpose

Specifies the password needed to connect to the Proxy Server.

Valid Values

`String`

where:

`String`

specifies the password to use to connect to the Proxy Server. Contact your system administrator to obtain your password.

Notes

- When proxy mode is disabled (`ProxyMode=0`), the Proxy Password option is ignored.
- Proxy Password is required only when the proxy server has been configured to require authentication.

Default

Empty string

GUI Tab

General tab

See Also

- Using IP addresses

- Proxy Host
- Proxy Mode
- Proxy Port
- Proxy User

Proxy Port

Attribute

ProxyPort (PXPT)

Purpose

Specifies the port number where the Proxy Server is listening for HTTP requests.

Valid Values

port_name

where:

port_name

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

Notes

- When proxy mode is disabled (`ProxyMode=0`), the Proxy Port option is ignored.

Default

`0`

GUI Tab

General tab

See Also

- Using IP addresses
- Proxy Host

- Proxy Mode
- Proxy Password
- Proxy User

Proxy User

Attribute

ProxyUser (PXU)

Purpose

Specifies the user name needed to connect to the Proxy Server.

Valid Values

The default user ID that is used to connect to the Proxy Server.

Notes

- When proxy mode is disabled (`ProxyMode=0`), the Proxy User option is ignored.
- Proxy User is required only when the proxy server has been configured to require authentication.

Default

`Empty string`

GUI Tab

General tab

See Also

- Using IP addresses
- Proxy Host
- Proxy Mode
- Proxy Password
- Proxy Port

Query Timeout

Attribute

QueryTimeout (QT)

Purpose

The number of seconds for the default query timeout for all statements that are created by a connection. To override the value set by this connection option for an individual statement, set a different value in the SQL_ATTR_QUERY_TIMEOUT statement attribute on the SQLSetStmtAttr() function.

Valid Values

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

where:

x

is a positive integer that specifies a number of seconds.

Behavior

If set to `-1`, the query does not time out. The driver silently ignores the SQL_ATTR_QUERY_TIMEOUT attribute.

If set to `0`, the query does not time out, but the driver responds to the SQL_ATTR_QUERY_TIMEOUT attribute.

If set to *x*, all queries time out after the specified number of seconds unless the application overrides this value by setting the SQL_ATTR_QUERY_TIMEOUT attribute.

Default

`0`

GUI Tab

Advanced tab

Record Delimiter

Attribute

BulkLoadRecordDelimiter (BLRD)

Purpose

Specifies the character that the driver will use to delimit the record entries in a bulk load data file.

Valid Values

x

where:

x

is any printable character.

For simplicity, avoid using a value that can be in the data, including all alphanumeric characters, the dash(-), the colon(:), the period (.), the forward slash (/), the space character, the single quote (') and the double quote ("). You can use some of these characters as delimiters if all of the data in the file is contained within double quotes.

Notes

- The Bulk Load Record Delimiter character must be different from the Bulk Load Field Delimiter.

Default

None

GUI Tab

Bulk tab

Report Codepage Conversion Errors

Attribute

ReportCodepageConversionErrors (RCCE)

Purpose

Specifies how the driver handles code page conversion errors that occur when a character cannot be converted from one character set to another.

An error message or warning can occur if an ODBC call causes a conversion error, or if an error occurs during code page conversions to and from the database or to and from the application. The error or warning generated is `Code page conversion error encountered`. In the case of parameter data conversion errors, the driver adds the following sentence: `Error in parameter x`, where *x* is the parameter number. The standard rules for returning specific row and column errors for bulk operations apply.

Valid Values

`0` | `1` | `2`

Behavior

If set to `0` (Ignore Errors), the driver substitutes 0x1A for each character that cannot be converted and does not return a warning or error.

If set to `1` (Return Error), the driver returns an error instead of substituting 0x1A for unconverted characters.

If set to `2` (Return Warning), the driver substitutes 0x1A for each character that cannot be converted and returns a warning.

Default

`0` (Ignore Errors)

GUI Tab

Advanced tab

Socket Idle Time

Attribute

SocketIdleTimeCheckInterval (SITCI)

Purpose

Specifies the interval of time, in seconds, at which the driver checks the connection for activity when Keep Connection Active is enabled (`KeepConnectionActive=1`). If no activity has been detected during this period, the driver issues a lightweight query (`Select 0`) to the database to maintain the connection. This

functionality provides a method to keep open connections to data sources that use Azure SQL Gateway, such as Azure SQL Database and Azure Synapse Analytics, during periods of inactivity.

Valid Values

x

where:

x

is the interval of time, in seconds, at which the driver checks the connection for activity.

Default

`1500` (seconds)

GUI Tab

Advanced tab

See Also

- Keep Connection Active

SSLLibName

Attribute

SSLLibName (SLN)

Purpose

The absolute path for the OpenSSL library file containing the TLS/SSL library to be used by the data source or connection when TLS/SSL is enabled. The TLS/SSL library contains the implementations of TLS/SSL protocols the driver uses for data encryption.

This option allows you to designate a different TLS/SSL library if you encounter issues with the default version or want to use a library that you provide. Common issues that require designating a different library include security vulnerabilities with specific libraries or compatibility issues with your server or application.

Valid Values

absolute_path\openssl_filename

where:

absolute_path

is the absolute path to where the OpenSSL file is located

openssl_filename

is the name of the OpenSSL library file containing the TLS/SSL Library to be used by your data source or connection.

Example

C:\Program Files\Progress\DataDirect\ODBC\Drivers\ddopenssl30.dll

Notes

- **Warning**: If you are distributing the driver with your application, you must prevent your end users from setting the value for the SSLLibName option. The SSLLibName option provides a method for you to specify an OpenSSL library file used for SSL encryption. However, if exposed, the option can be used to specify files that execute malicious or undesirable code. Refer to "Security best practices for ODBC applications" in the *Progress DataDirect for ODBC Drivers Reference* for more information.
- The value specified for this option should be an absolute path to a mounted drive.
- The OpenSSL library files provided by Progress combine the cryptographic and TLS/SSL libraries into a single file; therefore, when your drivers are using a Progress library file, the values specified for the CryptoLibName and SSLLibName options should be the same. For non-Progress library files, the libraries may use separate files, which would require unique values to be specified.
- This option can be used to designate OpenSSL libraries not installed by the product; however, the drivers are only certified against libraries provided by Progress.

Default

No default value

GUI Tab

The value for this option is specified as an option-value pair in the Extended Options field on the Advanced tab. For example:

SSLLibName=C:\Program Files\Progress\DataDirect\ODBC\Drivers\ddopenssl30.dll;

See also

- Advanced tab
- CryptoLibName

TCP Keep Alive

Attribute

KeepAlive (KA)

Purpose

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

Valid Values

`0` | `1`

Behavior

If set to `0` (Disabled), the driver does not enable TCPKeepAlive.

If set to `1` (Enabled), the driver enables TCPKeepAlive.

Default

`0` (Disabled)

Notes

- This option differs from Keep Connection Active (KeepConnectionActive) in that it maintains only the TCP socket, while Keep Connections Active maintains the connection to the database or Azure SQL Gateway. If you have a database or gateway timeout, you will need to enable Keep Connection Active to prevent your connection from terminating during idle periods.

GUI Tab

Advanced tab

See Also

- Keep Connection Active

Trust Store

Attribute

Truststore (TS)

Purpose

Specifies either the path and file name of the truststore file or the contents of the TLS/SSL certificates to be used when TLS/SSL is enabled and server authentication is used.

Valid Values.

*truststore_directory**filename*|data://-----BEGIN CERTIFICATE-----*certificate_content*-----END CERTIFICATE-----

where:

truststore_directory

is the path to the directory where the truststore file is located.

filename

is the file name of the truststore file.

certificate_content

is the content of the TLS/SSL certificate.

Notes

- **Warning**: If you are distributing the driver with your application, you must prevent your end users from setting the value for the Truststore option. The Truststore option provides a method for you to specify a truststore file used for TLS/SSL encryption. However, if exposed, the option can be used to specify files that execute malicious or undesirable code. Refer to "Security best practices for ODBC applications" in the *Progress DataDirect for ODBC Drivers Reference* for more information.
- The value specified for this option should be an absolute path to a mounted drive.
- If you do not specify the path to the directory that contains the truststore file, the current directory is used for authentication.

- The keystore and truststore files may be the same file.
- When specifying content for multiple certificates, secify the content of each certificate between `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`. For example:

  ```
  -----BEGIN CERTIFICATE-----certificatecontent1-----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----certificatecontent2-----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----certificatecontent3-----END CERTIFICATE-----
  ```

  Note that the number of dashes (`-----`) must be the same before and after both `BEGIN CERTIFICATE` and `END CERTIFICATE`.
- When specifying the certificate content for authentication, do not specify the truststore password. Since the truststore file is not required to be stored on the disk when the certificate content is specified directly, the driver need not unlock its contents.
- The Trust Store field on the Driver setup dialog supports content up to 8192 characters in length. For specifying certificate content longer than 8192 characters, edit the registry and manually add the entry to the DSN.
- On Windows platforms, if the required certificates are available in the Windows certificate store, the Trust Store and Truststore Password options need not be used.

Default

No default value

GUI Tab

Security tab

Trust Store Password

Attribute

TruststorePassword (TSP)

Purpose

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

Valid Values

`truststore_password`

where:

truststore_password

is a valid password for the truststore file.

Notes

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

Default

None

GUI Tab

Security tab

Use Snapshot Transactions

Attribute

SnapshotSerializable (SS)

Purpose

Allows your application to use the snapshot isolation level if your Microsoft SQL Server database is configured for Snapshot isolation. Supported only for Microsoft SQL Server 2005 and higher.

See "Using the Snapshot isolation level" for details about using the snapshot isolation level.

Valid Values

`0` | `1`

Behavior

When set to `1` (Enabled) and your application has the transaction isolation level set to serializable, the application uses the snapshot isolation level.

When set to `0` (Disabled) and your application has the transaction isolation level set to serializable, the application uses the serializable isolation level.

This option is useful for existing applications that set the isolation level to serializable. Using Snapshot Transactions in this case allows you to change to the snapshot isolation level with no or minimum code

changes. If developing a new application, you can code it to set the connection attribute SQL_COPT_SS_TXN_ISOLATION to the value SQL_TXN_SS_SNAPSHOT.

Notes

- This connection option can affect performance.

Default

`0` (Disabled)

GUI Tab

Advanced tab

See Also

- Using the Snapshot isolation level
- Performance considerations

User Name

Attribute

LogonID (UID)

Purpose

Specifies one of the following identifiers used for authentication:

- The default user ID that is used to connect to your database.
- The object (principal) ID of the Azure SQL logical server. This value is used when authenticating the service principal user with Entra ID (`AuthenticationMethod=36`).
- The client ID for the user-assigned managed identity used for authentication. This is an optional value used when authenticating with Managed Identity authentication (`AuthenticationMethod=37`). Note that this value is required only if you have multiple user-assigned identities. If you have only one user-assigned identity, the driver will connect without specifying a value for this option.

Your ODBC application may override this value or you may override it in the logon dialog box or connection string.

Valid Values

user_id|*principal_id*|*client_id*|

where:

user_id

is a valid user ID with permissions to access the database.

principal_id

is the object (principal) ID used for service principal user authentication with Entra ID.

client_id

is the client ID for the user-assigned managed identity used for authentication.

Default

None

GUI Tab

Security tab

Validate Server Certificate

Attribute

ValidateServerCertificate (VSC)

Purpose

Determines whether the driver validates the certificate that is sent by the database server when TLS/SSL encryption is enabled. When using TLS/SSL server authentication, any certificate sent by the server must be issued by a trusted Certificate Authority (CA). Allowing the driver to trust any certificate 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.

Truststore information is specified using the Trust Store and Trust Store Password options.

Valid Values

`0` | `1`

Behavior

If set to `1` (Enabled), 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 Host Name In Certificate option is specified, the driver also validates the certificate using a host name. The Host Name In Certificate option 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 `0` (Disabled), the driver does not validate the certificate that is sent by the database server. The driver ignores any truststore information specified by the Trust Store and Trust Store Password options.

Notes

- This option is ignored when `EncryptionMethod=8` (Strict). The driver always validates the server certificate when using strict connection encryption.

Default

`1` (Enabled)

GUI Tab

Security tab

Workstation ID

Attribute

WorkstationID (WSID)

Purpose

The workstation ID that is used by the client.

Valid Values

`string`

where:

string

is the workstation ID.

Default

None

GUI Tab

Advanced tab

XML Describe Type

Attribute

XMLDescribeType (XDT)

Purpose

The SQL data type that is returned by SQLGetTypeInfo for the XML data type.

See "Using the XML data type" for further information about the XML data type.

Valid Values

`-4` | `-10`

Behavior

If set to `-4` (SQL_LONGVARBINARY), the driver uses the description SQL_LONGVARBINARY for columns that are defined as the XML data type.

If set to `-10` (SQL_WLONGVARCHAR), the driver uses the description SQL_WLONGVARCHAR for columns that are defined as the XML data type.

Default

`-10`

GUI Tab

Advanced tab

See Also

Using the XML data type