Progress DataDirect for ODBC for MongoDB
User's Guide

Release 8.1.0

Copyright

Visit the following page online to see Progress Software Corporation’s current Product Documentation Copyright Notice/Trademark Legend: https://www.progress.com/legal/documentation-copyright.

Updated: 2025/07/30

Table of Contents

Welcome to the Progress DataDirect for ODBC for MongoDB Driver......9
What's new in this release?....10
Driver requirements....14
Migrating schema maps and native files to 8.1....17
Installing and setting up the driver (Windows)....18
Installing and setting up the driver (UNIX/Linux)....20
Connection string examples....22
User ID and password authentication....23
No authentication....24
Kerberos authentication....25
LDAP authentication....26
TLS/SSL encryption....27
Replica set failover....29
MongoDB Atlas....30
Microsoft Azure Cosmos DB for MongoDB....31
Data types....32
Default mapping of columns with inconsistent native data types....33
Mapping objects to tables....34
Normalized view....35
Flattened view....44
Mixed view....44
Driver specifications....52
Additional information....53
Troubleshooting....53
Contacting Technical Support....53

Tutorials....55
The Example application....55
Power BI (Windows only)....57
Tableau (Windows only)....57
Microsoft Excel (Windows only)....58

Configuring and connecting to data sources....61
Environment settings....62
Windows environment variables....62
UNIX/Linux environment variables....62
UTF-16 applications on UNIX and Linux....65
Configuring data sources with the Configuration Manager....65

Generating connection strings with the Configuration Manager....66
Using a connection string....67
Additional configuration methods for UNIX and Linux....68
Configuration through the system information (odbc.ini) file....68
DSN-less connections....71
File data sources....73
Testing connections and queries with the Configuration Manager....73
Password Encryption Tool (UNIX/Linux only)....74
Using a logon dialog box....75
Authentication....75
User ID and password authentication....75
LDAP authentication....77
Kerberos authentication....78
TLS/SSL encryption....79
TLS/SSL server authentication....79
TLS/SSL client authentication....80
Supported keystores and truststores....81
Proxy server....82
MongoDB Atlas clusters....82
Replica set failover for write operations....84
Performance considerations....85

Using the SQL engine server....87

Configuring the SQL engine server using the Configuration Manager....88
Stopping the SQL engine server using the Configuration Manager....90
Configuring Broker mode for the SQL engine server on UNIX/Linux....90
Configuring server mode using Java options....92
Stopping the SQL engine server....93
Configuring Java logging for the SQL engine server....94

Additional features and functionality....95

MongoDB sharding support....96
Using identifiers....96
Parameter metadata support....98
Binding parameter markers....99
MongoDB views....100
Local views....100
MinKey and MaxKey values....100
Packet logging....101

Connection option descriptions....105

Application Using Threads....112
Array Normalization Threshold....113

Authentication Method....114
Authentication Database....115
Broker Idle Timeout....115
Broker Ping Interval....116
Broker Port Number....117
Column Discovery Sample Size....117
Create Map....118
Crypto Protocol Version....119
Data Source Name....120
Database Name....120
Description....121
Enable DNS Lookup....121
Encryption Method....122
Extended Options....123
Fetch Size....124
Flatten Array Base....125
Host Name....125
Host Name in Certificate....126
Initialization String....126
JSON Columns....127
JVM Arguments....128
JVM Classpath....129
JVM Path....129
Key Password....130
Keystore....130
Keystore Password....131
Keyword Conflict Suffix....132
Leading Underscore Replacement....133
Legacy Virtual Keys....133
Log Config File....134
Max Connections Per Server....135
Min Varchar Size....135
Network Message Compressors....136
Password....137
Port Number....137
Proxy Host....138
Proxy Password....138
Proxy Port....139
Proxy User....139
Qualify Normalized Names....140
Read Only....141
Read Preference....141
Refresh Schema....142
Replica Set Name....143
Report Codepage Conversion Errors....144

Result Memory Size........144
Schema Filter........145
Schema Format........147
Schema Map........148
Server Idle Timeout........149
Server Launch Timeout........149
Server Port Number........150
Service Principal Name........151
Special Char Behavior........152
SQL Engine Mode........152
SQL Service........153
String Truncation Method for Writes........154
Timestamp Format........154
Transaction Mode........155
Truststore........156
Truststore Password........156
Uppercase Identifiers........157
User........158
Validate Server Certificate........158
Varchar Threshold........159

Supported SQL statements and extensions........161

Create View........162
Delete........163
Drop View........164
Insert........165
Refresh Map (EXT)........166
Reload Map (EXT)........167
Select........167
    Select Clause........169
Update........179
SQL Expressions........180
    Column Names........180
    Literals........180
    Operators........183
    Functions........186
    Conditions........186
Subqueries........187
    IN Predicate........188
    EXISTS Predicate........188
    UNIQUE Predicate........188
    Correlated Subqueries........189
Custom function escapes........190
    CAST_TO_NATIVE function escape........190

1

Welcome to the Progress DataDirect for ODBC for MongoDB Driver

The Progress® DataDirect® *for* ODBC™ for MongoDB™ driver supports SQL to select data from MongoDB data sources. Some insert, update, and delete capabilities are also supported. The driver translates the SQL statements provided by an application, enabling you to leverage your knowledge of SQL. Additionally, the driver creates a relational schema of your native MongoDB data to support SQL access to MongoDB's flexible schema data model. The relational schema created by the driver is written to a configuration file that can be shared among client machines accessing a common data store.

Once the driver has generated a relational schema, the relationships among objects can be reported through the following metadata methods: SQLColumns, SQLForeignKeys, SQLGetTypeInfo, SQLPrimaryKeys, SQLSpecialColumns, SQLStatistics, and SQLTables. Furthermore, when performing joins, the driver leverages data relationships in MongoDB collections, minimizing the amount of data that needs to be fetched over the network.

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.

For details, see the following topics:

- What's new in this release?
- Driver requirements
- Migrating schema maps and native files to 8.1
- Installing and setting up the driver (Windows)

- Installing and setting up the driver (UNIX/Linux)
- Connection string examples
- Data types
- Mapping objects to tables
- Driver specifications
- Additional information
- Troubleshooting
- Contacting Technical Support

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 8.1.0 GA

- **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 the TLSv1.3 cryptographic protocol. See Crypto Protocol Version on page 119 for details.
  - The driver has been enhanced to allow you to configure whether the driver inserts String values that exceed the column length defined in the relational schema. You can configure this behavior using the new String Truncation Method For Writes (StringTruncationMethodForWrites) options. See String Truncation Method for Writes on page 154 for details.
  - The driver has been enhanced to allow you to increase the default length of VARCHAR columns to increase the size of data values you can insert with the driver. You can configure the new Min Varchar Size (MinVarcharSize) option to specify minimum default length of fields mapped to VARCHAR. See Min Varchar Size on page 135 for details.
  - The driver has been enhanced to allow you to specify the format in which the driver renders the MongoDB composite timestamp. This allows you to specify the format of the MongoDB composite timestamp that is most appropriate for your use case. You can configure this driver behavior using the new Timestamp Format (TimestampFormat) option. See Timestamp Format on page 154 for details.
  - The SQL engine server has been enhanced to automatically start and stop the service as needed. When the new Broker mode is enabled (`SQLEngineMode=3`), the SQL engine operates in an external Java process that is monitored by the driver's Broker. The Broker eliminates the need to explicitly start and stop the service, as well as more efficiently manage memory and resource consumption. See Using the SQL engine server on page 87 for details.

- The driver has been enhanced to support replacing the internal mapping files at connection. When Create Map (CreateMap) is set to `1` (ForceNew), the driver replaces the schema map files specified by the Schema Map (SchemaMap) option with newly generated files at the same location. In addition, if you are upgrading from driver version 8.0 to 8.1, the driver retains a copy of the 8.0 version of the schema map files as a backup when generating the 8.1 version of the files. See Create Map on page 118 for details.
- The driver has been enhanced with the new Array Normalization Threshold (ArrayNormalizationThreshold) connection option. Array Normalization Threshold allows you to specify the length of arrays (in elements) at which the driver begins to normalize arrays to child tables when generating a flattened view. This provides you with a method to control the size and focus of your parent table when encountering large arrays or nested arrays. See Array Normalization Threshold on page 113 for details
- The driver has been enhanced to support data compression for all messages passed between the client and server. Data compression can significantly reduce network traffic, which, in turn, can lower data transfer costs for cloud services. See Network Message Compressors on page 136 for details.
- The driver has been enhanced to support MongoDB views created in the database. MongoDB views are discovered during the sampling process and mapped using the same schema format used for your collections. See MongoDB views on page 100 for details.
- The driver has been enhanced to support generating legacy virtual keys for newly-discovered nested objects. For driver versions earlier than 8.1, the driver used legacy virtual keys (unique identifiers) as a foreign key to associate the child table back to the parent. When the new LegacyVirtualKeys (LegacyVirtualKeys) option is set to `1` (true), the driver generates legacy virtual keys in the `*object_name*_GENERATED_ID` column in new and migrated schemas. This functionality allows you to maintain consistent column naming for driver generated virtual key columns when migrating normalized schema maps from the 8.0 format. See Legacy Virtual Keys on page 133 for details.
- The driver has been enhanced to support connections to Microsoft Azure Cosmos DB for MongoDB. See Microsoft Azure Cosmos DB for MongoDB on page 31 for details.
- The driver has been enhanced to support the JavaScript and Regex data types. See Data types on page 32 for details.

- **Changed behavior**
  - The driver no longer compresses messages passed between the client and the server by default. Additional research has uncovered that network compression might not provide the best performance in all environments. As a result, the default behavior of the Network Message Compressors (NetworkMessageCompressors) option has been changed to `none`. See Network Message Compressors on page 136 for details.
  - The behaviors of valid values for the Create Map (CreateMap) option have been changed as follows:
    - If set to 0 (No), the driver uses the current group of internal files specified by the Schema Map option. If the files do not exist, the connection fails.
    - If set to 2 (NotExist), the driver uses the current group of internal files specified by the Schema Map (SchemaMap) option. If the files do not exist, the driver creates them.

    See Create Map on page 118 for details.
  - The SchemaDefinition attribute has been added as an alias for the Schema Map (SchemaMap) option. This change allows users to continue using their configurations for earlier versions of the driver until they are able update their connection settings. Note that SchemaDefinition is a deprecated attribute and may not be supported in future versions of the driver. See Schema Map on page 148 for details.

Changes for 8.1.0 GA

- **Enhancements**
  - The driver has been enhanced to support generating legacy virtual keys for new nested objects. Legacy virtual keys are unique identifiers used to handle joins in version 8.0 and earlier of the driver. When the Legacy Virtual Keys (LegacyVirtualKeys) option is set to `1` (true), the driver generates legacy virtual keys in the *object_name*_GENERATED_ID column for new and migrated schemas. This functionality provides a method to continue using legacy virtual keys, if you prefer. See Legacy Virtual Keys on page 133 for details.
  - The normalization algorithm has been upgraded to improve sampling performance and optimize the generation of tables in the relational view. In addition, the new Schema Format (SchemaFormat) connection option allows you to determine to which relational view the driver maps data, including normalized, mixed, and flattened views. See Mapping objects to tables on page 34 for details.
  - The driver supports migrating schema maps and internal files created with the 8.0 version of the driver so that they can be used by the 8.1 driver. By migrating these files, you can continue to execute the same SQL statements that you did with the 8.0 driver while still leveraging the advantages of the 8.1 driver. See Migrating schema maps and native files to 8.1 on page 17 for details.
  - The driver has been enhanced to support LDAP (Lightweight Directory Access Protocol) authentication. See LDAP authentication on page 77 for details.
  - The new Schema Filter (SchemaFilter) connection option allows you to specify the database and collections pairs for which you want the driver to fetch metadata. Configuring this option can significantly improve connection times by limiting the collections for which metadata is fetched to only those that are required by your application. See Schema Filter on page 145 for details.
  - The new Qualify Normalized Names (QualifyNormalizedNames) option allows you to determine whether names of relational child-tables normalized from arrays, objects, subdocuments are prefixed with the collection name and names of any parent objects. See Qualify Normalized Names on page 140 for details.
  - The new Special Char Behavior (SpecialCharBehavior) option allows you to control how the driver handles the mapping of native identifiers containing characters that would require them to be quoted in SQL statements. This option provides a method to choose to continue using identifiers that require quotation marks or for the driver to modify affected identifier names so that quotation marks are not required. See Special Char Behavior on page 152 for details.
  - The new JSON columns (JSONColumns) option allows you to determine whether the driver exposes documents and arrays embedded within a collection as JSON formatted fields in addition to the individual collection and array elements being mapped as fields when using flattened mapping. Note that Querying JSON values can be an expensive operation that could negatively impact performance; therefore, you should only query JSON values when necessary. See JSON Columns on page 127 for details.
  - The new Flatten Array Base (FlattenArrayBase) option allows you to specify the starting ordinal value appended to column names for flattened arrays. When flattening arrays, column names are appended with an underscore and the ordinal value (*<array_name>_<ordinal_location>*). This option allows you to determine whether the first ordinal value in the series is either a `0` or a `1`. See Flatten Array Base on page 125 for details.
  - The driver has been enhanced to support replica set failover for write operations. This feature can be enabled by specifying your replica set name using the new Replica Set Name (ReplicaSetName) connection option. SeeReplica set failover for write operations on page 84 and Replica Set Name on page 143 for details.
  - The driver has been enhanced to support connections to MongoDB Atlas clusters using a DNS seed list. Instead of specifying connection information for individual nodes, the driver allows you to specify the name of the DNS SRV record using the Host Name (HostName) option. The driver then uses a DNS lookup to discover the member nodes in the cluster to which it can connect. You can control whether

driver performs a DNS lookup using the new Enable DNS Lookup (EnableDNSLookup) option. See MongoDB Atlas clusters on page 82 for details.

- On Windows, the driver now includes the MongoDB Configuration Manager for quick configuration and testing of your driver in a web browser. The tool allows you to:
  - Configure data sources and connection strings
  - Test connectivity for your data sources and connection strings
  - Execute SQL commands for testing purposes

  For details, see Configuring data sources with the Configuration Manager on page 65, Generating connection strings with the Configuration Manager on page 66, and Testing connections and queries with the Configuration Manager on page 73.

- The driver has been enhanced with the new Authentication Database (AuthenticationDatabase) connection option, which provides a method to specify the database in which the user id was created for user id and password authentication (`AuthenticationMethod=userIdPassword`). This allows you to explicitly select a set of credentials when the same user ID was created on multiple databases. See Authentication Database on page 115 for details.
- The driver has been enhanced to support the SCRAM-SHA-256 authentication method for user ID and password authentication. When user ID and password authentication is enabled (`AuthenticationMethod=UserIDPassword`), the driver detects and uses the most secure method supported by the server. See Authentication Method on page 114 for details.
- The driver has been enhanced to use proxy server settings defined in the JVM system option by default. If no proxy settings are defined in the connection string or data source, the driver will attempt to use the values of the `http.proxyHost` and `http.proxyPort` JVM system option to connect to the database. See Proxy server on page 82 for details.
- The driver has been enhanced to allow you to limit sampling to only new collections when refreshing the schema map. This provides for quicker processing times if you only want to map new collections or if existing collections are unchanged. You can limit sampling to only new collections by specifying the `NEW` option in a `Refresh Map` statement. See Refresh Map (EXT) on page 166 for details.
- The driver has been enhanced to support the native Decimal128 data type, which maps to the Decimal ODBC type by default. See Data types on page 32 for details.
- The driver has added support for the MinKey and MaxKey special values. See MinKey and MaxKey values on page 100 for details.
- The driver has 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 CAST_TO_NATIVE function escape has been introduced to select or insert a value of a specific native type. This can be particularly useful when MongoDB has inconsistent native types for a given field. Currently, CAST_TO_NATIVE can only be used with the ObjectID type in SELECT statement filters and literal INSERT values. See CAST_TO_NATIVE function escape on page 190 for details.

- **Changed behavior**
  - The default mapping behavior of the driver has been changed from generating a normalized view of data to one that is a mixture of normalized and flattened views. The mixed view reduces the number of tables generated while providing a more intuitive data model. You can configure the mapping behavior using the new Schema Format (SchemaFormat) option. For details see Mixed view on page 44 and Schema Format on page 147 for details.

- The Schema Tool and manual customizations of the schema map are not currently supported. However, you can control certain aspects of relational data model generated by the driver using mapping connection options. . See Connection option descriptions on page 105 for details.
- The Create Database (CreateDB) connection option has been replaced by the Create Map (CreateMap) option. The valid values and behavior are identical for both options. See Create Map on page 118 for details.
- The Database (Database) connection options has been replaced by the Database Name (DatabaseName) option. The valid values and behavior are identical for both options. See Database Name on page 120 for details.
- The User Name (LogonID) connection option has been replaced with the User (User) option. The valid values and behavior are identical for both options. See User on page 158 for details.
- The Login Timeout (LoginTimeout) is not currently supported. To specify the login time out behavior for an individual connection, set a value in the SQL_ATTR_LOGIN_TIMEOUT connection attribute using the SQLSetConnectAttr() function.
- The Config Options (ConfigOptions) connection option is no longer supported. As a result, the driver has been enhanced to support setting each of the configuration options formerly supported by Config Options as standalone connection options. See Connection option descriptions on page 105 for details.
- The following Config Options are no longer supported:
  - DefaultVarcharSize
  - MaxVarcharSize
  - MinVarcharSize

  To determine the default length of VARCHAR fields, the driver multiplies the largest discovered field by 1.5. For example, if the largest detected field has a length of 100 characters, the driver sets the default length to 150 characters.
- The driver does not currently support the IBM System z Linux platform.
- Support has ended for database versions earlier than MongoDB 3.6.
- Support for has ended for the following 32-bit platforms:
  - HP-UX PA RISC
  - Solaris x86
  - Solaris SPARC

  JVM support for 32-bit versions of HP-UX PA RISC, Solaris x86, and Solaris SPARC ended with Oracle JDK 7, OpenJDK 7, IBM SDK 7. As a result, the driver can no longer receive updates for these platforms due to the driver's JVM requirements.

Driver requirements

Data source and platform requirements

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

Java requirements

- The driver requires a Java Virtual Machine (JVM) that is Java SE 8 or higher. JVM support includes Oracle JDK, OpenJDK, and IBM SDK (Java) distributions.
- For 32-bit drivers, a 32-bit Java Virtual Machine (JVM) is required. For 64-bit drivers, a 64-bit Java Virtual Machine (JVM) is required.
- For Windows, you must set the PATH environment variable to the directory containing the `jvm.dll` for your JVM.
- For UNIX/Linux, you must set the library path environment variable of your operating system to the directory containing your JVM's `libjvm.so` file and that directory's parent directory.

AIX requirements for 32-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
- IBM SDK, JAVA Technology Edition
- Before you can use the driver, you must set the `LIBPATH` environment variable to include the paths containing the `libjvm.so` library and the `libnio.so` library, which are installed in a subdirectory of your Java Development Kit (JDK). For example, you would add the following paths for Java 8 installed in the `/usr` directory:

  ```
  :/usr/java8/jre/lib/ppc/classic:/usr/java8/jre/lib/ppc
  ```

  In this example, `/usr/java8/jre/lib/ppc/classic` is the location of `libjvm.so`, while `/usr/java8/jre/lib/ppc` is the location of `libnio.so`.

Note: The driver is compiled using the `-brtl` loader option. Your application must support runtime linking functionality.

AIX requirements for 64-bit drivers

- IBM POWER Processor
- An application compatible with components that were built using Visual Age C++ version 6.0.0.0 and the AIX native threading model
- IBM SDK, JAVA Technology Edition
- Before you can use the driver, you must set the `LIBPATH` environment variable to include the paths containing the `libjvm.so` library and the `libnio.so` library, which are installed in a subdirectory of your Java Development Kit (JDK). For example, you would add the following paths for Java 8 installed in the `/usr` directory:

  ```
  :/usr/java8_64/jre/lib/ppc64/classic:/usr/java8_64/jre/lib/ppc64
  ```

  In this example, `/usr/java8_64/jre/lib/ppc64/classic` is the location of `libjvm.so`, while `/usr/java8_64/jre/lib/ppc64` is the location of `libnio.so`.

Note: The driver is compiled using the `-brtl` loader option. Your application must support runtime linking functionality.

HP-UX requirements for 32-bit drivers

- Intel Itanium II (IPF) processor
- 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)
- You must set the LD_PRELOAD environment variable to the fully qualified path to the `libjvm.so` from your JVM installation.

Note:

- Do not link with the –lc linker option.

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)
- You must set the LD_PRELOAD environment variable to the fully qualified path to the `libjvm.so` of your JVM installation.

Note: Do not link with the –lc linker option.

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).

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 Sun Studio 11, C++ compiler version 5.8 and the Solaris native (kernel) threading model
- For x64: An application compatible with components that were built using Sun C++ Compiler version 5.8 and the Solaris native (kernel) threading model

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.

Migrating schema maps and native files to 8.1

The driver supports migrating schema maps created with the 8.0 version of the driver so that they can be used by the 8.1 driver. By migrating these files, you can continue to execute the same SQL statements that you did with the 8.0 driver while leveraging the advantages of the 8.1 driver. Note that new objects discovered by the driver are mapped using the behavior of the 8.1 driver. See "Mapping objects to tables" for details.

Note: You can configure the driver to use the 8.0 name format for virtual foreign key columns for new objects by setting Legacy Virtual Keys (LegacyVirtualKeys) option to `1` (true). See "Legacy Virtual Keys" for details.

To migrate an existing 8.0 schema map and internal files, using the Schema Map (SchemaMap) option, specify the fully qualified path to the configuration file used for the 8.0 driver and set SchemaFormat to `NormalizeAll`. At connection, the driver migrates the content of your 8.0 schema map files so that they are compatible with the 8.1 driver. In addition, the driver creates copies of the original files in the same directory, if you want to continue using the 8.0 driver while you complete your upgrade. The copies of the original files take the following form:

```
<existing_file>-<file_version>.<extension>
```

For example, version 2 of a file named `myserver.config` would be renamed:

```
myserver-2.config
```

Note: The versions of the configuration files and internal files are not synched. Therefore, the files will likely have different numbers appended to the them for the 8.0 versions of the files.

Using migrated schema maps and internal files have the following limitation:

- The driver supports only normalized relational views (`SchemaFormat=NormalizeAll`) for migrated schema maps and internal files.
- The driver does not support migrating maps for flattened views; however, the flattened views generated for 8.0 and 8.1 are very similar and should support many of the same SQL statements. You can further modify your flattened view to support SQL statements for earlier versions by configuring the Flatten Array Base (FlattenArrayBase) and JSON Columns (JSON Columns) options.

See also

Mapping objects to tables on page 34
Legacy Virtual Keys on page 133
Mapping objects to tables on page 34
Flatten Array Base on page 125
JSON Columns on page 127

Installing and setting up the driver (Windows)

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

To begin accessing data with the driver:

1. Install the driver:
   a) After downloading the product, unzip the installer files to a temporary directory.
   b) From the installer directory, run the appropriate installer file to start the installer. The installer file takes the following form:

      `PROGRESS_DATADIRECT_ODBC_nn_WIN_xx_INSTALL.exe`

   c) Follow the prompts to complete installation.

   Note:

   The installer program supports multiple installation methods, including command-line and silent installations. For detailed instructions, refer to the *Progress DataDirect for ODBC Drivers Installation Guide*.

   ---

2. Before you can use your driver, you must set the PATH environment variable to include the path of the `jvm.dll` file of your Java™ Virtual Machine (JVM).
3. To configure the driver using the ODBC Administrator (GUI), start the ODBC Administrator from the Progress DataDirect program group. The GUI dialog allows you to configure the data source definitions in the Windows Registry or generate connection strings.

   Note: The Windows driver also supports using connection strings to connect to your service. For more information, see "Using a connection string."

   ---

4. Select 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 your 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 your driver and click **Finish** to display the driver Setup dialog box.
   - **File DSN**: Configuring a new file data source using the File DSN tab is not currently supported with the configuration manager.
5. The **Connection** tab of the Configuration Manager opens in a window. Provide values for the following essential connection options; then, click **Apply**:

For all connections:

- **Data Source Name**: Type a string that identifies this data source configuration, such as `Projects`.
- **Description**: Type an optional long description of a data source name, such as `My Development Projects`.
- **Host Name**: Type the name or the IP address of the server to which you want to connect. For example, `myserver`.
- **Port Number**: (Optional) Type the port number of the server listener. The default is `27017`.
- **Database**: (Optional) Type the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.

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

For user User ID and password authentication:

- **Authentication Database**: (Recommended) Type the database in which your user ID was created. This value allows you to explicitly select a set of credentials and permissions when the same user ID was created in multiple databases on the server.
- **User**: Type the user name that is used to connect to a MongoDB database. For example, `jsmith`.
- **Password**: Type the password that is used to connect to the instance.

Note: User and Password connection options can also be specified in the logon dialog box or passed by your application.

Note: The driver supports a number of authentication methods. See Authentication on page 75 for more information.

6. Set the values for any additional connection options that you want to configure. To view more options, click on the tabs on the dialog. See the following resources for additional information on optional features and functionality:

   - Connection string examples on page 22 provides connection string examples that can be used to configure common functionality and features. The options and values described in this section apply to all configuration methods.
   - Connection option descriptions on page 105 provides a complete list of supported options by functionality.
   - Configuring data sources with the Configuration Manager on page 65 guides you through using the GUI to configure the driver.
   - Performance considerations on page 85 describes connection options that affect performance, along with recommended settings.

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

7. Click **Test Connect** to attempt to connect to the data source using the connection options.

8. The logon dialog appears. If not already specified, update the following fields; then, click **OK**.
   - **Host Name**: Type the name or the IP address of the server to which you want to connect.
   - **User**: Type the user name that is used to connect to a MongoDB database.
   - **Password**: Type the password that is used to connect to the instance.

   Note: The information you enter in the logon dialog box during a test connect is not saved.

9. If the test was successful, the window displays a confirmation message.
10. Click **OK** to close the setup dialog. 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 Configuration Manager to modify your data source, or you can override these defaults by connecting to the data source using a connection string with alternate values.
11. Connect to your server and begin accessing data with your applications, BI tools, database tools, and more. To help you get started, the following resources guide you through accessing data with some common tools:
    - Example Application: The example application allows you to test connect, execute SQL statements, and practice using the ODBC API right out of the box.
    - Power BI: Power BI is a business intelligence software program that allows you to generate analytics and visualized representations of your data.
    - Tableau: Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data.
    - Microsoft Excel: Excel is a spreadsheet tool that allows you to connect, view tables, and execute SQL statements against your data.
    - Connection option descriptions on page 105: This section describes the syntax used for SQL statements supported by the driver. You can modify and use the provided examples for your application or tool.

This completes the deployment of the driver.

See also

Using a connection string on page 67

Installing and setting up the driver (UNIX/Linux)

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

To begin accessing data with the driver:

1. Install the driver:
   a) After downloading the product, extract the contents of the product file.
   b) From the installer directory, run the installer's binary file to start the installer. The file for the installer program takes the following form:

   ```
   PROGRESS_DATADIRECT_ODBC_nn_platform_xx_INSTALL.bin
   ```

   c) Follow the prompts to complete installation.

The installer program supports multiple installation methods, including command-line and silent installations. For detailed instructions, refer to the *Progress DataDirect for ODBC Drivers Installation Guide*.

2. Configure the environment variables:

   a) Check your permissions. You must log in as a user with full r/w/x permissions recursively on the entire product installation directory.

   b) 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`. Executing the setup script:

      - Sets the ODBCINI environment variable to point to the path from the root directory to the system information file where your data source resides. For details, see "ODBCINI."
      - Sets the library path environment variable for your operating system to include the directory containing your JVM's `libjvm.so` [a] file. The library path environment variables are as follows:
        - `LD_LIBRARY_PATH` on Solaris, Linux, and HP-UX
        - `LIBPATH` on AIX
      - Sets the library path environment variable for your operating system to include the directory containing your JVM's `libjvm.so` [`sl` | a] file. The library path environment variables are as follows:
        - `LD_LIBRARY_PATH` on Solaris, Linux, and HP-UX
        - `LIBPATH` on AIX

   c) For HP-UX, you also must set the `LD_PRELOAD` environment variable to the fully qualified path of the `libjvm.so`.

3. Configure the driver using one of the following methods:

   - **odbc.ini file**: You can begin using the driver immediately by editing the `odbc.ini` file in the installation directory with a text editor. The following demonstrates a data source definition with the basic options for user ID and password authentication.

```
[ODBC Data Sources]
MongoDB=DataDirect 8.1 MongoDB


[MongoDB]
Driver=<install_dir>/lib/xxmongodb81.yy
...
Description=My MongoDB Data Source
...
AuthenticationDatabase=mydb2
...
DatabaseName=mydb
...
HostName=myserver
...
User=jsmith
...
Password=secret
...
```

   See Configuration through the system information (odbc.ini) file on page 68 for more information.

Note: The User 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.

- **Connection string**: The driver also supports using connection strings for DSN (data source name), File DSN, or DSN-less connections. See Using a connection string, DSN-less connections, for more information. For examples, see Connection string examples. See Generating connection strings with the Configuration Manager for information on generating connection strings using the Configuration Manager.

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

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

   - Connection string examples provides connection string examples that can be used to configure common functionality and features. You can modify and combine these examples to create a string that best suites your environment.

   Note: The options and values described in "Connection string examples" apply to all configuration methods.

   - Connection option descriptions provides a complete list of supported options by functionality.
   - Performance Considerations describes connection options that affect performance, along with recommended settings.

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

   - Example Application: The example application is a command-line tool that allows you to test connect, execute SQL statements, and practice using the ODBC API in environments that do not support GUIs.
   - Supported SQL statements and extensions on page 161: This section describes the syntax used for SQL statements supported by the driver. You can modify and use the provided examples for your application or tool.

This completes the deployment of the driver.

See also

ODBCINI on page 63
Connection string examples on page 22

Connection string examples

ODBC provides a method for specifying connection information via a connection string and the SQLDriverConnect API. This section provides examples of connection strings configured to use common features and functionality. You can modify and/or combine these examples to create a connection string for your environment.

In addition to the connection strings for DSN-less connections demonstrated in this section, the driver supports DSN and File DSN connection strings. See "Using a connection string" for syntax and detailed information for supported connection string types.

Note: The options and values described in this section apply to all configuration methods.

See also

Using a connection string on page 67

User ID and password authentication

This string includes the options used to connect using basic user name and password authentication.

Note: The strings demonstrated in this section use the DSN-less format. For additional formats, see Using a connection string on page 67.

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=auth_db;DatabaseName=database;
  HostName=hostname;PortNumber=port;User=user_name;Password=password;
  [attribute=value[;...]];
```

where:

auth_db

(recommended) specifies the database in which your user ID was created. This value allows you to explicitly select a set of credentials and permissions when the same user ID was created in multiple databases on the server.

Note: We recommend specifying a value for this option when using user ID and password authentication (`AuthenticationMethod=0`) to ensure that the correct permissions are used for your connection.

---

database

(optional) specifies the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.

Note: This value is case-insensitive if you have access privileges to query the list of databases on the server. If you do not have access, this value is case-sensitive.

---

hostname

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

port

specifies the port number of the server listener. The default is `27017`.

user_name

specifies the user name that is used to connect to the MongoDB database. For example, `jsmith`.

password

specifies the password used to connect to your MongoDB database.

attribute=value

specifies connection option settings. Multiple option attributes are separated by a semi-colon.

Note: The User 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.

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

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=mydb2;DatabaseName=mydb;
  HostName=myserver;PortNumber=27017;User=jsmith;Password=secret;
```

See also

User ID and password authentication on page 75
Connection option descriptions on page 105

No authentication

This string includes the options used to connect using no authentication.

Note: The strings demonstrated in this section use the DSN-less format. For additional formats, see Using a connection string on page 67.

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationMethod=None;DatabaseName=database;
  HostName=hostname;PortNumber=port;[attribute=value[;...]];
```

where:

database

(optional) specifies the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.

Note: This value is case-insensitive if you have access privileges to query the list of databases on the server. If you do not have access, this value is case-sensitive.

---

hostname

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

port

(optional) specifies the port number of the server listener. The default is `27017`.

attribute=value

specifies connection option settings. Multiple option attributes are separated by a semi-colon.

The following example connection string includes the options required for connecting with no authentication:

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=mydb2;DatabaseName=mydb;
       HostName=myserver;PortNumber=27017;User=jsmith;Password=secret;
```

See also

Connection option descriptions on page 105

Kerberos authentication

This string includes the options you may need to connect with Kerberos authentication.

Note: The strings demonstrated in this section use the DSN-less format. For additional formats, see "Using a connection string."

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

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationMethod=4;DatabaseName=database;
  HostName=hostname;PortNumber=port;ServicePrincipalName=principal_name;
  User=user_name;[attribute=value[;...]];
```

where:

database

(optional) specifies the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.

Note: This value is case-insensitive if you have access privileges to query the list of databases on the server. If you do not have access, this value is case-sensitive.

---

hostname

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

port

specifies the port number of the server listener. The default is `27017`.

user_name

(optional) specifies the user name that is used to connect to the MongoDB database. For example, `jsmith`.

principal_name

(optional) if the default value built by the driver does not match the service principal name registered with the KDC, specify the three-part service principal name. By default, the driver builds the ServicePrincipalName by concatenating the service name `mongodb`, the fully qualified domain name

(FQDN) as specified with the HostName property, and the default realm name as specified in the Kerberos configuration file.

attribute=value

specifies connection option settings. Multiple option attributes are separated by a semi-colon.

Note: The User 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.

The following example connection string includes the options required for connecting with LDAP authentication:

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=4;DatabaseName=mydb;
       HostName=myserver;PortNumber=27017;User=jsmith;
```

See also

Using a connection string on page 67
Connection option descriptions on page 105

LDAP authentication

This string includes the options you may need to connect with LDAP authentication.

Note: The strings demonstrated in this section use the DSN-less format. For additional formats, see "Using a connection string."

This string includes the options used to connect with LDAP authentication.

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationMethod=17;DatabaseName=database;
  HostName=hostname;PortNumber=port;User=user_name;Password=password;
  [attribute=value[;...]];
```

where:

database

(optional) specifies the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.

Note: This value is case-insensitive if you have access privileges to query the list of databases on the server. If you do not have access, this value is case-sensitive.

hostname

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

port

specifies the port number of the server listener. The default is `27017`.

user_name

(optional) specifies the user name that is used to connect to the MongoDB database. For example, `jsmith`.

password

specifies the password used to connect to your MongoDB database.

attribute=value

specifies connection option settings. Multiple option attributes are separated by a semi-colon.

Important: When LDAP authentication is enabled, credentials are passed in clear text. Therefore, you should use LDAP authentication only on servers that are configured for TLS/SSL encryption.

Note: The User 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.

The following example connection string includes the options required for connecting with LDAP authentication:

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=17;DatabaseName=mydb;
       HostName=myserver;PortNumber=27017;User=jsmith;Password=secret;
```

See also

Using a connection string on page 67
Connection option descriptions on page 105

TLS/SSL encryption

This string includes the options you may need to connect with TLS/SSL encryption and user ID and password authentication. For more information, see "TLS/SSL data encryption."

Note: The strings demonstrated in this section use the DSN-less format. For additional formats, see "Using a connection string."

This string includes the options used to connect through a proxy server with authentication.

```
DRIVER=DataDirect 8.1 MongoDB;CryptoProtocolVersion=protocol;
  DatabaseName=database;EncryptionMethod=1;HostName=hostname;
  HostNameInCertificate=host;KeyPassword=key_password;KeyStore=key_store;
  PortNumber=port;TrustStore=trust_store;TrustStorePassword=ts_password;
  ValidateServerCertificate=vsc_value;User=user_name;Password=password;
  [attribute=value[;...]];
```

where:

protocol

(optional) specifies a cryptographic protocol or comma-separated list of cryptographic protocols that can be used when SSL is enabled. For example, `TLSv1.3,TLSv1.2`.

database

(optional) specifies the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.

Note: This value is case-insensitive if you have access privileges to query the list of databases on the server. If you do not have access, this value is case-sensitive.

hostname

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

host

(optional) specifies the host name to be used to validate the certificate. The HostNameInCertificate property provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

key_password

(optional) specifies the password of the keystore file, if your database server is configured for SSL client authentication. Alternatively, you can set the corresponding Java system property javax.net.ssl.keyStorePassword, in lieu of this option.

key_store

(optional) specifies the location of the keystore file, if your database server is configured for SSL client authentication. Alternatively, you can set the corresponding Java system property, javax.net.ssl.keyStore, in lieu of this property.

trust_store

(optional) specifies the location of the truststore file used for SSL server authentication. Alternatively, you can set the corresponding Java system property, javax.net.ssl.trustStore, in lieu of this property.

ts_password

(optional) specifies the password of the truststore file used for SSL server authentication. Alternatively, you can set the corresponding Java system property, javax.net.ssl.trustStorePassword, in lieu of this option.

vsc_value

(optional) specify true to validate certificates sent by the database server.

port

specifies the port number of the server listener. The default is `27017`.

user_name

specifies the user name that is used to connect to the MongoDB database. For example, `jsmith`.

password

specifies the password used to connect to your MongoDB database.

attribute=value

specifies connection option settings. Multiple option attributes are separated by a semi-colon.

Note: The User 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.

The following example connection string includes the minimum options required for connecting with data encryption, and user ID and password authentication:

```
DRIVER=DataDirect 8.1 MongoDB;DatabaseName=mydb;EncryptionMethod=1;HostName=myserver;
       PortNumber=27017;User=jsmith;Password=secret;
```

See also

TLS/SSL encryption on page 79
Using a connection string on page 67
Connection option descriptions on page 105

Replica set failover

This string includes the options used to enable replica set failover for write operations with no authentication.

Note: The strings demonstrated in this section use the DSN-less format. For additional formats, see "Using a connection string."

This string includes the options used to connect through a proxy server with authentication.

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationMethod=15;DatabaseName=database;
  HostName=hostname;PortNumber=port;ReplicaSetName=replica_set;
  [attribute=value[;...]];
```

where:

database

(optional) specifies the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.

Note: This value is case-insensitive if you have access privileges to query the list of databases on the server. If you do not have access, this value is case-sensitive.

---

hostname

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

port

specifies the port number of the server listener. The default is `27017`.

replica_set

specifies the name of the replica set against which you want to execute write operations.

attribute=value

specifies connection option settings. Multiple option attributes are separated by a semi-colon.

The following example connection string includes the options used to enable replica set failover for write operations with no authentication:

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationMethod=15;DatabaseName=mydb;
       HostName=myserver;PortNumber=27017;ReplicaSetName=MyReplicaSet;
```

See also

Using a connection string on page 67
Connection option descriptions on page 105
Replica set failover for write operations on page 84

MongoDB Atlas

This string includes the options used to connect to a MongoDB Atlas cluster.

Note: The strings demonstrated in this section use the DSN-less format. For additional formats, see "Using a connection string."

This string includes the options used to connect through a proxy server with authentication.

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=auth_db;
  DatabaseName=database;EncryptionMethod=1;HostName=hostname;PortNumber=port;
  User=user_name;Password=password;[attribute=value[;...]];;
```

where:

auth_db

(recommended) specifies the database in which your user ID was created. This value allows you to explicitly select a set of credentials and permissions when the same user ID was created in multiple databases on the server.

Note: We recommend specifying a value for this option when using user ID and password authentication (`AuthenticationMethod=0`) to ensure that the correct permissions are used for your connection.

database

(optional) specifies the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.

Note: This value is case-insensitive if you have access privileges to query the list of databases on the server. If you do not have access, this value is case-sensitive.

hostname

specifies the domain name of the MongoDB Atlas cluster to which you want to connect. At connection, the driver performs a DNS lookup to identify the member nodes of the domain, then connects to an available node.

port

specifies the port number of the server listener. The default is `27017`.

user_name

specifies the user name that is used to connect to the MongoDB database. For example, `jsmith`.

password

specifies the password used to connect to your MongoDB database.

attribute=value

specifies connection option settings. Multiple option attributes are separated by a semi-colon.

Note: The User 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.

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

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=mydb2;DatabaseName=mydb;
  EncryptionMethod=1;HostName=//myaltas.mongodb.net;PortNumber=27017;User=jsmith;
  Password=secret;
```

See also

Using a connection string on page 67
Connection option descriptions on page 105
MongoDB Atlas clusters on page 82

Microsoft Azure Cosmos DB for MongoDB

This string includes the options used to connect to a Microsoft Azure Cosmos DB for MongoDB. Connection strings for Azure Cosmos DB for MongoDB are similar to standard MongoDB connections, but require TLS encryption and authentication to be enabled.

Note: The strings demonstrated in this section use the DSN-less format. For additional formats, see "Using a connection string."

This string includes the options used to connect through a proxy server with authentication.

```
DRIVER=DataDirect 8.1 MongoDB;DatabaseName=database;EncryptionMethod=1;
  HostName=hostname;PortNumber=port;ValidateServerCertificate=0;User=user_name;
  Password=password;[attribute=value[;...]];
```

where:

database

(optional) specifies the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.

Note: This value is case-insensitive if you have access privileges to query the list of databases on the server. If you do not have access, this value is case-sensitive.

hostname

specifies the name or the IP address of the Azure Cosmos DB for MongoDB server to which you want to connect. For example, `myCosmosdbServer`.

port

specifies the port number of the server listener. This value is typically `10255` for Azure Cosmos DB for MongoDB connections.

user_name

(optional) specifies the your Azure Cosmos DB account name. For example, `jsmith`.

password

(optional) specifies the password used to connect to your Azure Cosmos DB account.

attribute=value

specifies connection option settings. Multiple option attributes are separated by a semi-colon.

Note: The User 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.

The following example connection string includes the basic properties used for connecting to a Azure Cosmos DB database.

```
DRIVER=DataDirect 8.1 MongoDB;DatabaseName=mydb;EncryptionMethod=1;
       HostName=myCosmosdbServer;PortNumber=10255;ValidateServerCertificate=0;
       User=jsmith;Password=secret;
```

See also

Using a connection string on page 67
Connection option descriptions on page 105

Data types

The following table lists native data types supported by the driver and how they are mapped to ODBC data types.

Table 1: MongoDB Data Types

MongoDB Data Type	ODBC Data Type
ARRAY	SQL_LONGVARCHAR
BIGINT	SQL_BIGINT
BINDATA	SQL_VARBINARY
BOOLEAN	SQL_BIT
CHAR	SQL_CHAR
DATE	SQL_TYPE_TIMESTAMP
DECIMAL128	SQL_DECIMAL
DOUBLE	SQL_DOUBLE
INTEGER	SQL_INTEGER
JAVASCRIPT	SQL_VARCHAR
LONGVARCHAR	SQL_LONGVARCHAR
OBJECT	SQL_LONGVARCHAR
OBJECTID	SQL_VARCHAR
REGEX[1]	SQL_VARCHAR
STRING	SQL_VARCHAR

Default mapping of columns with inconsistent native data types

Due to the flexibility of the MongoDB schema data model, your native data sources may not enforce consistent data types. To ensure data integrity when mapping to a relational model, the driver describes columns with inconsistent native data types as a single SQL data type. The driver determines which SQL type to use based on the combination of native types detected when sampling data. The following table lists combinations of MongoDB data types and their default mapping for ODBC.

Note: In some cases, a value with a specific native type can be concealed or obscured because the driver describes a field with inconsistent native types as a column with a single SQL type. The CAST_TO_NATIVE function escape allows users to send a value as it is defined in the MongoDB database, rather than how it is described in the relational model of the data. Currently, CAST_TO_NATIVE can only be used with the ObjectID type in SELECT statement filters and literal INSERT values. See "CAST_TO_NATIVE function escape" for details.

[1] The driver supports only the literal format for the Regex data type. Therefore, when executing an Insert or Select statement, Regex values must be specified using the literal format.

Table 2: Default Mapping for Columns Containing Inconsistent MongoDB Data Types

MongoDB Data Types	ODBC Data Type
Bigint and Integer	SQL_BIGINT
Double and Integer	SQL_DOUBLE
All other combinations	SQL_WVARCHAR or WLONGVARCHAR[2]

See also

CAST_TO_NATIVE function escape on page 190

Mapping objects to tables

Data mapping describes how elements are mapped between two distinct data models. To support SQL access to a MongoDB database, the MongoDB database must be mapped to a relational schema. The driver creates the relational schema by mapping MongoDB data in a normalized, flattened, or mixed view of the data.

- **Normalized view**: MongoDB collections are normalized into sets of parent-child tables. The child tables correspond to complex types, such as arrays and embedded documents (or subdocuments), and have a foreign key relationship to the parent table. For more information, see "Normalized view."
- **Flattened view**: MongoDB collections are flattened into single, relational tables. All fields, including those that may comprise complex types, are organized as columns within the relational table. For more information, see "Flattened view."
- **Mixed view**: The driver generates an optimized view of the data in which MongoDB collections are mapped to a normalized view that flattens certain objects into columns in the parent table. For example, subdocuments are mapped as columns in the parent table, while most arrays and nested subdocuments are mapped to separate child tables. See "Mixed view" for more information.

The driver generates the relational view upon the initial connection to a data source. By default, the driver generates a mixed view of relational data. You can determine the relational view using the Schema Format (`SchemaFormat`) option.

At connection, the relational schema is written to the schema map configuration file. This configuration file may be shared by multiple users. Note that users must have appropriate privileges to the exposed collections to access them.

Note: The names of relational tables demonstrated in this section are generated based on the default behavior of the driver. You can modify how the driver generates table names using the Qualify Normalized Names (`QualifyNormalizedNames`) option.

See also

Schema Format on page 147
Qualify Normalized Names on page 140

[2] Whether columns are mapped as either WVARCHAR or WLONGVARCHAR depends on the size of the data of the sampled rows.

Normalized view

When your native data is normalized, each MongoDB collection is decompounded into a set of parent-child tables. Fields containing simple types are mapped as columns in a parent table, while complex types, such as subdocuments and arrays, are mapped as child tables. Child tables share a foreign key relationship with their parent table.

For example, the collection `residents` contains the array `vehicles` and fields in the `address` document (or object). The collection's JSON structure can be rendered as follows:

```
{"_id": "ajx363",
 "name": "Sydney Smith",
 "address": {"street": "101 Main Street", "city": "Raleigh", "state": "NC"},
 "county": "Wake",
 "vehicles": ["car", "boat"]
}

{"_id": "tzn525",
 "name": "Cora Welch",
 "address": {"street": "191 First Street", "city": "Chapel Hill", "state": "NC"},
 "county": "Orange",
 "vehicles": ["scooter", "truck"]
}
```

The collection has been decompounded into separate but related tables. The normalization of the `residents` collection yields one parent table and two child tables. In the parent table, the `_id` field is mapped as a primary key column, and fields with other simple types are mapped as relational columns. The parent table adopts the name `RESIDENTS` and takes the following form:

Table 3: RESIDENTS

_ID (PK)	NAME	COUNTY
ajx363	Sydney Smith	Wake
tzn525	Cora Welch	Orange

The information in the `vehicles` array is mapped to the following child table:

Table 4: RESIDENTS_VEHICLES

RESIDENTS_ID (FK)	POSITION	VEHICLES
ajx363	0	car
ajx363	1	boat
tzn525	0	scooter
tzn525	1	truck

The mapping of the `vehicles` array to the `RESIDENTS_VEHICLES` table is handled as follows:

- The table name, RESIDENTS_VEHICLES, is derived from the name of the parent table and the array.
- To establish a foreign key relationship with the parent table, a column is generated based on the _ID field using the following naming convention:

  <parent_table>_ID

  For example, RESIDENTS_ID.
- The primary key of the child table is a composite key formed by the primary key of the parent table (RESIDENTS_ID) combined with the positional information contained in the POSITION column.
- Fields in the object are mapped to columns in the child table. For example, the vehicles field becomes the VEHICLES column.

The information in the address object maps to the following child table:

Table 5: RESIDENTS_ADDRESS

RESIDENTS_ID (PK & FK)	STREET	CITY	STATE
ajx363	101 Main Street	Raleigh	NC
tzn525	191 First Street	Chapel Hill	NC

The mapping of the address object to the RESIDENTS ADDRESS table is handled as follows:

- The table name, RESIDENTS_ADDRESS, is derived from the parent table and the name of the object.
- To establish a foreign key relationship with the parent table, a column is generated based on the _ID field using the following naming convention:

  <parent_table>_ID

  For example, RESIDENTS_ID.
- Fields in the object are mapped to columns in the child table. For example, the city field becomes the CITY column.

Nested complex types (normalized view)

The following examples illustrate a number of ways the normalization of complex types are handled.

A subdocument nested in a subdocument

In this example, the subdocument `location` contains the subdocuments `city` and `state` in the `employee` collection:

```
{"_id": "pdn313",
 "name": "Charlotte",
 "manager": {"name": "Robert", "department": "Development",
            "location": {"city": "Tulsa", "state": "OK"}}
}

{"_id": "gkx136",
 "name": "Benjamin",
 "manager": {"name": "Michael", "department": "Quality Assurance",
            "location": {"city": "Dallas", "state": "TX"}}
}
```

The information in the `employee` collection would be mapped to the `EMPLOYEE` parent table, and the `EMPLOYEE_MANAGER` and the `EMPLOYEE_LOCATION` child tables. The `EMPLOYEE` table would be derived from the simple types `_id` and `name` as follows:

Table 6: EMPLOYEE

_ID	NAME
pdn313	Charlotte
gkx136	Benjamin

The information from the `manager` object is mapped to the following child table:

Table 7: EMPLOYEE_MANAGER

	NAME	DEPARTMENT
pdn313	Robert	Development
gkx136	Michael	Quality Assurance

The mapping of the `manager` object to the `EMPLOYEE_MANAGER` table is handled as follows:

- The table name, `EMPLOYEE_MANAGER`, is derived from the name of the parent table and the object.
- To establish a foreign key relationship with the parent table, a column is generated based on the `_ID` field using the following naming convention:

  ```
  <parent_table>_ID
  ```

  For example, `EMPLOYEE_ID`.
- The nested subdocuments are mapped to columns in the child table. For example, the `name` field is mapped to the `NAME` column.

The information in the `location` object is mapped to the following child table:

Table 8: EMPLOYEE_LOCATION

EMPLOYEE_ID (PK & FK)	CITY	STATE
pdn313	Tulsa	OK
gkx136	Dallas	TX

The mapping of the `location` object to the `EMPLOYEE_LOCATION` table is handled as follows:

- The table name, `EMPLOYEE_LOCATION`, is derived from the name of the parent table and the object.
- The `id_` field is mapped as `EMPLOYEE_ID` to establish a foreign key relationship to the parent table.
- Nested subdocuments are mapped to columns. In this example, the `city` and `state` subdocuments, which indicate the location of an employee's manager, are mapped as `CITY` and `STATE` columns.

Subdocuments nested in an array

In the collection `contacts`, the `addresses` array contains the subdocuments: `label`, `street`, `city`, and `state`.

```
{"_id": "ajx456",
 "name": "Andrea",
 "addresses": [
              {"label": "Home", "street": "101 Main St", "zip": "27513"},
              {"label": "Work", "street": "303 Main St", "zip": "27560"}
              ]
}
```

The information in the `contacts` collection would be mapped to the `CONTACT` parent table, and the `CONTACTS_ADDRESSES` child table. The `CONTACTS` table would be derived from the simple types `_id` and `name` as follows:

Table 9: CONTACTS

_ID	NAME
ajx456	Andrea

The information in the `addresses` array maps to the following child table:

Table 10: CONTACTS_ADDRESSES

CONTACTS_ID (FK)	POSITION (PK)	LABEL	STREET	ZIP
ajx456	0	Home	101 Main St	27513
ajx456	1	Work	303 Main St	27560

The mapping of the `addresses` array to the `CONTACTS_ADDRESSES` tables is handled as follows:

- The table name, `CONTACTS_ADDRESSES`, is derived from the name of the parent table and array.
- To establish a foreign key relationship with the parent table, a column is generated based on the `_ID` field using the following naming convention:

  `<parent_table>_ID`

For example, CONTACTS_ID.

- The primary key of the child table is a composite key formed by the primary key of the parent table (CONTACTS_ID) combined with the positional information contained in the POSITION column.
- Subdocuments in an array are mapped as columns. For example, the label field is mapped to the LABEL column.

An array nested in a document

In a separate scenario, the manager document contains the emails array in the employee collection. The collection has the following JSON structure:

```
{"_id": "ajp211",
 "name": "Mark",
 "manager": {"name": "Cynthia",
           "emails": ["cynthia@email.com", "watsonc@email.com"]}
}
{"_id": "mpc393",
 "name": "Deborah",
 "manager": {"name": "Cynthia",
           "emails": ["cynthia@email.com", "watsonc@email.com"]}
}
{"_id": "dlm215",
 "name": "Jason",
 "manager": {"name": "Chris",
           "emails": ["chris@email.com", "cwright@email.com"]}
}
```

The information in the employee collection would be mapped to the EMPLOYEE parent table, and the EMPLOYEE_EMAILS and EMPLOYEE_MANAGER child tables. The EMPLOYEE table would be derived from the simple types _id and name as follows:

Table 11: EMPLOYEE

_ID	NAME
ajp211	Mark
mpc393	Deborah
dlm215	Jason

The information in the emails array is mapped to the following child table:

Table 12: EMPLOYEE_EMAILS

EMPLOYEE_ID (FK)	POSITION	EMAILS
ajp211	0	cynthia@email.com
ajp211	1	watsonc@email.com
mpc393	0	cynthia@email.com
mpc393	1	watsonc@email.com
dlm215	0	chris@email.com
dlm215	1	cwright@email.com

The mapping of the nested `emails` array to the `EMPLOYEE_EMAILS` table is handled as follows:

- The table name, `EMPLOYEE_EMAILS`, is derived from the name of the parent table and the array.
- To establish a foreign key relationship with the parent table, a column is generated based on the `_ID` field using the following naming convention:

  <parent_table>`_ID`

  For example, `EMPLOYEE_ID`.
- The primary key of the child table is a composite key formed by the primary key of the parent table (`EMPLOYEE_ID`) combined with the positional information contained in the `POSITION` column.
- Nested arrays are mapped as columns. For example, the `emails` array is mapped to the `EMAILS` column. Note that the emails in the following child table are those of the managers, but correspond to the employee identification number.

The information in the `names` subdocument is mapped to the following child table:

Table 13: EMPLOYEE_MANAGER

EMPLOYEE_ID (PK & FK)	NAME
ajp211	Cynthia
mpc393	Cynthia
dlm215	Chris

The mapping of the nested `names` subdocument to the `EMPLOYEE_MANAGER` table is handled as follows:

- The table name, `EMPLOYEE_MANAGER`, is derived from the name of the parent table and the object.
- To establish a foreign key relationship with the parent table, a column is generated based on the `_ID` field using the following naming convention:

  <parent_table>`_ID`

  For example, `EMPLOYEE_ID`.
- The nested `name` subdocument (the name of the manager) is mapped as a column. Note that the names in the following child table are those of the managers, but correspond to the employee identification number.

An array nested in an array

In the collection `offices`, the `departments` array contains an `employees` array.

```
{"_id": "au32",
 "city": "Bedford",
 "departments": [{"name": "Development",
                  "employees": [
                                 {"first": "Leslie", "last": "Jacobs"},
                                 {"first": "Emma", "last": "Alves"},
                                 {"first": "Brad", "last": "Jones"}
                                ]
                 },
                 {"name": "Human Resources",
                  "employees": [
                                 {"first": "Joseph", "last": "Lu"},
                                 {"first": "Margaret", "last": "Baker"},
                                 {"first": "Chetna", "last": "Campbell"}
                                ]
                 },
                 {"name": "IT",
                  "employees": [
                                 {"first": "Caroline", "last": "Evans"},
                                 {"first": "Markus", "last": "Campanella"},
                                 {"first": "Jennifer", "last": "Bradley"}
                                ]
                 }]
 }
{"_id": "xn44",
 "city": "Morrisville",
 "departments": [{"name": "Development",
                  "employees": [
                                 {"first": "Charles", "last": "Scott"},
                                 {"first": "Mary", "last": "Gonzales"},
                                 {"first": "Phil", "last": "McEnroe"}
                                ]
                 },
                 {"name": "Operations",
                  "employees": [
                                 {"first": "Rachel", "last": "Cullingford"},
                                 {"first": "Lance", "last": "Friedman"},
                                 {"first": "Amanda", "last": "Giachetti"}
                                ]
                 },
                 {"name": "IT",
                  "employees": [
                                 {"first": "Ingrid", "last": "Burkis"},
                                 {"first": "Catherine", "last": "Wheeler"},
                                 {"first": "Jacob", "last": "Williams"}
                                ]
                 }]
}
```

The information in the offices collection would be mapped to the OFFICES parent table, and the OFFICES_DEPARTMENTS and OFFICES_EMPLOYEES child tables. The OFFICES table would be derived from the simple types _id and city as follows:

Table 14: OFFICES

_ID (PK)	CITY
au32	Bedford
xn44	Morrisville

The information in the departments nested array is mapped to the following child table:

Table 15: OFFICES_DEPARTMENTS

OFFICES_ID (FK)	POSITION (PK)	NAME
au32	0	Development
au32	1	Human Resources
au32	2	IT
xn44	0	Development
xn44	1	Operations
xn44	2	IT

The mapping of the departments nested array to the OFFICES_DEPARTMENTS table is handled as follows:

- The table name, OFFICES_DEPARTMENTS, is derived from the name of the parent table and the array.
- To establish a foreign key relationship with the parent table, a column is generated based on the _ID field using the following naming convention:

  *<parent_table>*_ID

  For example, OFFICES_ID.
- The primary key of the child table is a composite key formed by the primary key of the parent table (OFFICES_ID) combined with the positional information contained in the POSITION column.
- Nested arrays are mapped as columns. For example, the name array is mapped to the NAME column.

The information in the employees array is mapped to the following child table:

Table 16: OFFICES_EMPLOYEES

	OFFICES_ DEPARTMENTS _POSITION	POSITION (PK)	FIRST	LAST
au32	0	0	Leslie	Jacobs
au32	0	1	Emma	Alves
au32	0	2	Brad	Jones
au32	1	0	Joseph	Lu
au32	1	1	Margaret	Baker
au32	1	2	Chetna	Campbell
au32	2	0	Caroline	Evans
au32	2	1	Markus	Campanella
au32	2	2	Jennifer	Bradley
xn44	0	0	Charles	Scott
xn44	0	1	Mary	Gonzales
xn44	0	2	Phil	McEnroe
xn44	1	0	Rachel	Cullingford
xn44	1	1	Lance	Friedman
xn44	1	2	Amanda	Giachetti
xn44	2	0	Ingrid	Burkis
xn44	2	1	Catherine	Wheeler
xn44	2	2	Jacob	Williams

The mapping of the nested `employees` array to the `OFFICES_EMPLOYEES` table is handled as follows:

- The table name, `OFFICES_EMPLOYEES`, is derived from the name of the parent table and the array.
- The `OFFICES_ID` and `OFFICES_DEPARTMENTS_POSITION` columns establish a foreign key relationship to the parent array based on the primary key from the `DEPARTMENTS` table.
- The primary key of the child table is a composite key formed by the primary key of the parent table (`OFFICES_ID`) combined with the positional information contained in the `POSITION` column.
- Nested arrays are mapped as columns. For example, the `first` array is mapped to the `FIRST` column.

Flattened view

The following example shows how the source data is flattened.

A data source has a collection called `employee` with the following structure:

```
{"_id": "pdn313",
 "name": "Charlotte",
 "manager": {"name": "Robert",
             "emails": ["bob@email.com", "robert@email.com"]
            }
}
```

The `employee` collection would be flattened into a relational table with the following structure:

_ID	NAME	MANAGER_NAME	MANAGER_EMAILS_1	MANAGER_EMAILS_2
pdn313	Charlotte	Robert	bob@email.com	robert@email.com

All fields are retained as separate columns in the resulting relational table. Simple types such as `_id` and `name` correspond directly to columns. In turn, subdocuments are flattened into columns using the `<objectname>_<fieldname>` pattern. Next, the `emails` array is flattened into two columns, using an extended version of the `<arrayname>_<arrayindex>` pattern: `<objectname>_<arrayname>_<arrayindex>`.

Note: As subdocuments or arrays are discovered at deeper and deeper levels, the `<objectname>_<fieldname>` or `<arrayname>_<arrayindex>` pattern is extended, for example, `<objectname>_<objectname>_<fieldname>`.

Note: To avoid creating very large tables, arrays containing twelve or more elements are normalized into child tables by default. You can configure this behavior using the Array Normalization Threshold (ArrayNormalizationThreshold) option.

See also

Array Normalization Threshold on page 113

Mixed view

By default, the driver generates a mixed-normalized view, which changes the composition of the relational tables and which objects are mapped to child tables.

In the following example, the collection `residents` contains the array `vehicles` and fields in the `address` document (or object). The collection's JSON structure can be rendered as follows:

```
{"_id": "ajx363",
 "name": "Sydney Smith",
 "address": {"street": "101 Main Street", "city": "Raleigh", "state": "NC"},
 "county": "Wake",
 "vehicles": ["car", "boat"]
}

{"_id": "tzn525",
 "name": "Cora Welch",
 "address": {"street": "191 First Street", "city": "Chapel Hill", "state": "NC"},
 "county": "Orange",
 "vehicles": ["scooter", "truck"]
}
```

In the mixed view, fields containing simple types are mapped to the parent table and nested complex types and arrays are mapped as child tables. Subdocuments with simple types are appended to the parent table. Therefore, normalization of the `residents` collection, produces a single table. The resulting table takes the following form:

Table 17: RESIDENTS

_ID (PK)	NAME	ADDRESS_ STREET	ADDRESS_ CITY	ADDRESS_ STATE	COUNTY	VEHICLES_ 1	VEHICLES_ 2
ajx363	Sydney Smith	101 Main Street	Raleigh	NC	Wake	car	boat
tzn525	Cora Welch	191 First Street	Chapel Hill	NC	Orange	scooter	truck

The mapping of the `residents` collection to the `RESIDENTS` table is handled as follows:

- Simple types are mapped to columns in the parent table as columns. For example, the `name` object is mapped to `NAME` column.
- Fields for the subdocument `address` are mapped as columns to the parent table. Column names for fields generated from a subdocument take the following form:

  <subdocument_name>_<field_name>

  For example, for the field `street` in the subdocument `address`, the resulting column name would be `ADDRESS_STREET`.
- Fields in the vehicles array are mapped as columns to the parent table. Column names for fields generated from an array take the following form:

  <array_name>_<oridinal_location>

  Important: In the mixed view, if the number of values in an array are uniform across the collection and are less than or equal to twelve per element, the array values are flattened into columns in the parent table (for example, `VEHICLES_1`, `VEHICLES_2`). If the number of values are not uniform or exceed twelve per element, the driver maps the array to a child table.

  For example, for the second value of the array `vehicles`, the resulting column name would be `VEHICLE_2`.

Nested complex types (mixed view)

The following examples illustrate a number of ways the driver maps complex types in a mixed representation of the relational schema.

A subdocument nested in a subdocument

In this example, the subdocument `location` contains the subdocuments `city` and `state` in the `employee` collection:

```
{"_id": "pdn313",
 "name": "Charlotte",
 "manager": {"name": "Robert", "department": "Development",
 "location": {"city": "Tulsa", "state": "OK"}}
}

{"_id": "gkx136",
 "name": "Benjamin",
 "manager": {"name": "Michael", "department": "Quality Assurance",
             "location": {"city": "Dallas", "state": "TX"}}
}
```

The information from the `employee` object is mapped to the following table:

Table 18: EMPLOYEE

_ID	NAME	MANAGER_ NAME	MANAGER_ DEPARTMENT	MANAGER_ LOCATION_CITY	MANAGER_ LOCATION_STATE
pdn313	Charlotte	Robert	Development	Tulsa	OK
gkx136	Benjamin	Michael	Quality Assurance	Dallas	TX

The mapping of the `employee` collection to the `EMPLOYEE` table is handled as follows:

- Simple types are mapped to columns in the parent table as columns. For example, the `name` object is mapped to `NAMES` column.
- Subdocuments nested in subdocuments are mapped as columns to the parent table using the following naming convention:

  <subdocument_name>_<nested_subdocument_name>_<field_name>

  For example, for the `location` subdocument nested in the `manager` subdocument, the `city` field would map to a column name of `MANAGER_LOCATION_CITY`.

Subdocuments nested in an array

In the collection `contacts`, the `addresses` array contains the subdocuments: `label`, `street`, and `zip`.

```
{"_id": "ajx456",
 "name": "Andrea",
 "addresses": [
    {"label": "Home", "street": "101 Main St", "zip": "27513"},
    {"label": "Work", "street": "303 Main St", "zip": "27560"}
             ]
}
```

In this example, the information from the contacts collection is mapped to the CONTACTS parent table and CONTACTS_ADDRESSES child table. The CONTACTS parent table is derived from the _id and name simple types, which are mapped to the _ID and NAMES column. The resulting parent table would take the following form:

Table 19: CONTACTS

_ID	NAME
ajx456	Andrea

The information from the ADDRESSES array is mapped to the following child table:

Table 20: CONTACTS_ADDRESSSES

CONTACTS_ID (FK)	POSITION (PK)	LABEL	STREET	ZIP
ajx456	0	Home	101 Main St	27513
ajx456	1	Work	303 Main St	27560

The mapping of the parent array, addresses, to the CONTACTS_ADDRESSES table is handled as follows:

- The table name, CONTACTS_ADDRESSES, is derived from the name of the array.
- To establish a foreign key relationship with the parent table, a column is generated based on the _ID field using the following naming convention:

  ```
  <parent_table>_ID
  ```

  For example, CONTACTS_ID.
- The primary key of the child table is a composite key formed by the primary key of the parent table (CONTACTS_ID) combined with the positional information contained in the POSITION column.
- Fields in the array are mapped to columns in the child table. For example, the label field becomes the LABEL column.

An array nested in a document

In this scenario, the manager document contains the emails array in the employee collection. The collection has the following JSON structure:

```
{"_id": "ajp211",
 "name": "Mark",
 "manager": {"name": "Cynthia",
             "emails": ["cynthia@email.com", "watsonc@email.com"]}
}
{"_id": "mpc393",
 "name": "Deborah",
 "manager": {"name": "Cynthia",
             "emails": ["cynthia@email.com", "watsonc@email.com"]}
}
{"_id": "dlm215",
 "name": "Jason",
 "manager": {"name": "Chris",
             "emails": ["chris@email.com"]}
}
```

The information from the employee collection is mapped to the EMPLOYEE parent table and the EMPLOYEE_EMAILS child table.

Important: In the mixed view, if the number of values in an array are uniform across the collection and are less than or equal to twelve per element, the array values are flattened into columns in the parent table (for example, EMAILS_1, EMAILS_2, EMAILS_3). If the number of values are not uniform or exceed twelve per element, the driver maps the array to a separate child table, as demonstrated by this example.

The resulting parent table takes the following form:

Table 21: EMPLOYEE

_ID	NAME	MANAGER_NAME
ajp211	Mark	Cynthia
mpc393	Deborah	Cynthia
dlm215	Jason	Chris

Mapping for the EMPLOYEE parent table is handled as follows:

- Simple types are mapped to columns in the parent table as columns. For example, the name object is mapped to NAME column.
- Fields for the subdocument manager are mapped as columns to the parent table. Column names for fields generated from a subdocument take the following form:

  <subdocument_name>_<field_name>

  For example, for the field name in the subdocument manager, the resulting column name would be MANAGER_NAME.

The information from the emails nested array is mapped to the following child table:

Table 22: EMPLOYEE_EMAILS

EMPLOYEE_ID (FK)	POSITION (PK)	EMAILS
ajp211	0	cynthia@email.com
ajp211	1	watsonc@email.com
mpc393	0	cynthia@email.com
mpc393	1	watsonc@email.com
dlm215	0	chris@email.com

The mapping of the emails array to the EMPLOYEE_EMAILS table is handled as follows:

- The table name is now derived from the name of the array. For example, EMAILS.

- To establish a foreign key relationship with the parent table, a column is generated based on the _ID field using the following naming convention:

  ```
  <parent_table>_ID
  ```

  For example, EMPLOYEE_ID.

- The primary key of the child table is a composite key formed by the primary key of the parent table (MANAGER_ID) combined with the positional information contained in the POSITION column.
- Nested arrays are mapped to the column. In this example, the emails array is mapped to the EMAILS column.

An array nested in an array

In the collection offices, the departments array contains an employees array.

```
{"_id": "au32",
 "city": "Bedford",
 "departments": [{"name": "Development",
                  "employees": [
                                {"first": "Leslie", "last": "Jacobs"},
                                {"first": "Emma", "last": "Alves"},
                                {"first": "Brad", "last": "Jones"}
                               ]
                  },
                  {"name": "Human Resources",
                   "employees": [
                                 {"first": "Joseph", "last": "Lu"},
                                 {"first": "Margaret", "last": "Baker"},
                                 {"first": "Chetna", "last": "Campbell"}
                                ]
                  },
                  {"name": "IT",
                   "employees": [
                                 {"first": "Caroline", "last": "Evans"},
                                 {"first": "Markus", "last": "Campanella"},
                                 {"first": "Jennifer", "last": "Bradley"}
                                 ]
                  }]
}
{"_id": "xn44",
 "city": "Morrisville",
 "departments": [{"name": "Development",
                  "employees": [
                                {"first": "Charles", "last": "Scott"},
                                {"first": "Mary", "last": "Gonzales"},
                                {"first": "Phil", "last": "McEnroe"}
                               ]
                 },
                 {"name": "Operations",
                  "employees": [
                                {"first": "Rachel", "last": "Cullingford"},
                                {"first": "Lance", "last": "Friedman"},
                                {"first": "Amanda", "last": "Giachetti"}
                               ]
                 },
                 {"name": "IT",
                  "employees": [
                                {"first": "Ingrid", "last": "Burkis"},
                                {"first": "Catherine", "last": "Wheeler"},
                                {"first": "Jacob", "last": "Williams"}
                               ]
                  }]
}
```

The information from the `offices` collection is mapped to the `OFFICES` parent table and the `OFFICES_DEPARTMENTS` and `OFFICES_EMPLOYEES` child tables. The `OFFICES` parent table is derived from the `_id` and `city` simple types. The `_id` field is mapped as the primary key column `_ID`, and the `city` field is mapped as the relational column `CITY`. The resulting `OFFICES` parent table takes the following form:

Table 23: OFFICES

_ID (PK)	CITY
au32	Bedford
xn44	Morrisville

The information in the `departments` parent array is mapped to the following child table:

Table 24: OFFICES_DEPARTMENTS

OFFICES_ID (FK)	POSITION (PK)	NAME
au32	0	Development
au32	1	Human Resources
au32	2	IT
xn44	0	Development
xn44	1	Operations
xn44	2	IT

The mapping of the `departments` parent array to the `OFFICES_DEPARTMENTS` child table is handled as follows:

- The table name is now derived from the name of the parent table and the array. For example, `OFFICES_DEPARTMENTS`.
- To establish a foreign key relationship with the parent table, a column is generated based on the `_ID` field using the following naming convention:

  ```
  <parent_table>_ID
  ```

  For example, `OFFICES_ID`.
- The primary key of the child table is a composite key formed by the primary key of the parent table (`OFFICES_ID`) combined with the positional information contained in the `POSITION` column.
- The fields in the parent array are mapped to columns. For example, the `name` array is mapped to the `NAME` column.

The information in the `employees` nested array is mapped to the following child table:

Table 25: OFFICES_EMPLOYEES

OFFICES_ID (FK)	OFFICES_DEPARTMENTS_POSITION	POSITION (PK)	FIRST	LAST
au32	0	0	Leslie	Jacobs
au32	0	1	Emma	Alves
au32	0	2	Brad	Jones
au32	1	0	Joseph	Lu
au32	1	1	Margaret	Baker
au32	1	2	Chetna	Campbell
au32	2	0	Caroline	Evans
au32	2	1	Markus	Campanella
au32	2	2	Jennifer	Bradley
xn44	0	0	Charles	Scott
xn44	0	1	Mary	Gonzales
xn44	0	2	Phil	McEnroe
xn44	1	0	Rachel	Cullingford
xn44	1	1	Lance	Friedman
xn44	1	2	Amanda	Giachetti
xn44	2	0	Ingrid	Burkis
xn44	2	1	Catherine	Wheeler
xn44	2	2	Jacob	Williams

The mapping of the `departments` array to the `OFFICES_DEPARTMENTS` child table is handled as follows:

- The table name is derived from the name of the parent table and the array. For example, `OFFICES_EMPLOYEES`.
- The foreign key, `OFFICES_ID`, reflects the name of the parent table.
- A foreign key relationship with the parent array is established using the `<parent_array>_POSITION` column. For example, `OFFICES_DEPARTMENTS_POSITION`.
- The primary key of the child table is a composite key formed by the primary key of the parent (`OFFICES_ID`), the positional information of the parent of the array (`OFFICES_DEPARTMENTS_POSITION`), and the positional information contained in the `POSITION` column.

- The fields in the nested array are mapped to columns in the table. For example, the `first` array is mapped to the `FIRST` column.

Driver specifications

This section describes the general functionality supported by the driver.

- **ODBC compliance**: The driver is compliant with the Open Database Connectivity (ODBC) 3.52 specification. The driver is ODBC core compliant and supports some Level 1 and Level 2 features.

  Refer to "ODBC API and scalar functions" in the *Progress DataDirect for ODBC Drivers Reference* for a list of supported API functions.

  The driver supports only the following Level 2 functions:

  - SQLColumnPrivileges
  - SQLDescribeParam
  - SQLForeignKeys
  - SQLPrimaryKeys
  - SQLProcedures
  - SQLTablePrivileges

- **Unicode support**: The driver is fully Unicode enabled. On UNIX and Linux platforms, the driver supports both UTF-8 and UTF-16. On Windows platforms, the driver supports UCS-2/UTF-16 only.

  Refer to "Internationalization, localization, and Unicode" in the *Progress DataDirect for ODBC Drivers Reference* for details.

- **Isolation and lock levels**: Because transactions are not supported, the driver supports only the isolation level 0 (read uncommitted).

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

- **Connections and statements supported**: The driver supports multiple connections and multiple statements per connection.

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.

Troubleshooting

The *Progress DataDirect for ODBC Drivers Reference* provides information on troubleshooting problems should they occur.

Refer to the "Troubleshooting" section in the *Progress DataDirect for ODBC Drivers Reference* for details.

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.

November 2021, 8.1.0 Release of Progress DataDirect for ODBC for MongoDB, Version 0001

2

Tutorials

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

For details, see the following topics:

- The Example application
- Power BI (Windows only)
- Tableau (Windows only)
- Microsoft Excel (Windows only)

The Example application

The driver installation includes an ODBC application called Example that can be used for:

- Testing any type of SQL statement
- Testing database connections
- Verifying your database environment

It can also be used to demonstrate ODBC function calls, including the following:

- SQLAllocHandle
- SQLBindCol
- SQLBindParameter
- SQLColAttribute
- SQLConnect
- SQLDescribeCol
- SQLDescribeParam
- SQLDisconnect
- SQLDriverConnect
- SQLExecDirect
- SQLFetch
- SQLFreeHandle
- SQLFreeStmt
- SQLGetDiagRec
- SQLGetInfo
- SQLNumResultCols
- SQLPrepare
- SQLSetEnvAttr
- SQLSetStmtAttr

The Example application can be built using the files located in the `\samples\examples` directory of the DataDirect for ODBC Drivers installation directory.

Note:

- For Windows, you can build the Windows app for ANSI and Unicode.
- For UNIX/Linux, instructions for building the Example application are contained inside the file `example.mak`, which can be read with a text editor.

To use the Example application:

1. After you have configured the data source, navigate to the `instal_dir\samples\example` directory.
2. Open the application using one of the following methods:
   - Running the application executable or binary:
     - On Windows, double-click the `Example.exe` file.
     - On UNIX/Linux, run the `example` application.
   - Executing a command-line argument. For example:
     - `example` *`connection_string`*
     - `example "DSN" "UID" "PWD"`
     - `example` *`connection_string`* `"`*`sql_command_1`*`" ["`*`sql_command_2`*`" ...]`

   Results: A command prompt opens.
3. Follow the prompts to enter your data source name, user name, and password. If successful, a SQL> prompt appears.
4. At the prompt, enter SQL statements to test your connection. For example:

   ```
   SELECT * FROM INFORMATION_SCHEMA.TABLES
   ```

The results of your query are displayed. If example is unable to connect, the appropriate error message is returned.

Power BI (Windows only)

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

1. Navigate to the `\tools\Power BI` subdirectory of the Progress DataDirect installation directory; then, locate the installation batch file `install.bat`.
2. Run the `install.bat` file. The following operations are executed by running the `install.bat` file:
    - The Power BI connector file, `DataDirectMongoDB.pqx`, is copied to the following directory.

      `%USERPROFILE%\Documents\Power BI Desktop\Custom Connectors`
    - The following Windows registry entry is updated.

      `HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Power BI Desktop\TrustedCertificateThumbprints`
3. Open the Power BI desktop application.
4. From the **Get Data** window, navigate to **Other > Progress DataDirect MongoDB Connector**.
5. Click **Connect**. Then, from the **Progress DataDirect MongoDB Connector** window, provide the following information. Then, click **OK**.
    - **Data Source:** Enter a name for the data source. For example, `MongoDB ODBC DSN`.
    - **SQL Statement:** If desired, provide a SQL command.
    - **Data Connectivity mode:**
        - Select **Import** to import data to Power BI.
        - Select **DirectQuery** to query live data. (For details, including limitations, refer to the Microsoft Power BI article Use DirectQuery in Power BI Desktop.)
6. Enter authentication information when prompted. Once connected, the **Navigator** window displays schema and table information.
7. Select and load tables. Then, prepare your Power BI dashboard as desired.

You have successfully accessed your data and are now ready to create reports with Power BI. For more information, refer to the Power BI product documentation at Power BI documentation.

Tableau (Windows only)

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

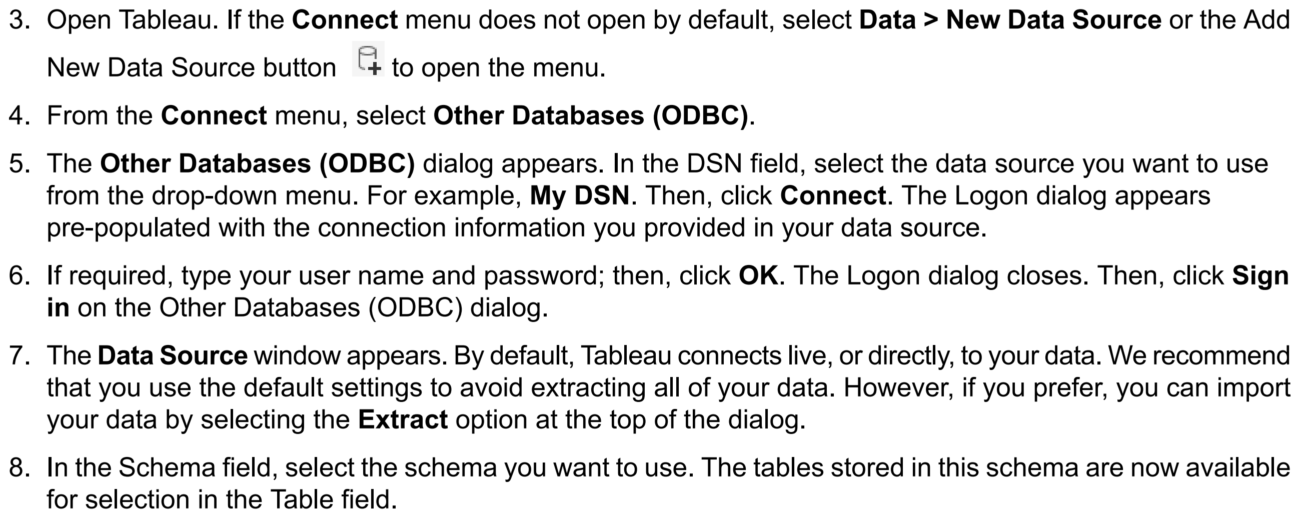
To use the driver to access data with Tableau:

1. Navigate to the `\tools\Tableau` subdirectory of the Progress DataDirect installation directory; then, locate the following Tableau data source file:

   `DataDirect MongoDB.tdc`

2. Copy the Tableau data source file into the following directory:

   `C:\Users\`*`user_name`*`\Documents\My Tableau Repository\Datasources`

3. Open Tableau. If the **Connect** menu does not open by default, select **Data > New Data Source** or the Add New Data Source button  to open the menu.
4. From the **Connect** menu, select **Other Databases (ODBC)**.
5. The **Other Databases (ODBC)** dialog appears. In the DSN field, select the data source you want to use from the drop-down menu. For example, **My DSN**. Then, click **Connect**. The Logon dialog appears pre-populated with the connection information you provided in your data source.
6. If required, type your user name and password; then, click **OK**. The Logon dialog closes. Then, click **Sign in** on the Other Databases (ODBC) dialog.
7. The **Data Source** window appears. By default, Tableau connects live, or directly, to your data. We recommend that you use the default settings to avoid extracting all of your data. However, if you prefer, you can import your data by selecting the **Extract** option at the top of the dialog.
8. In the Schema field, select the schema you want to use. The tables stored in this schema are now available for selection in the Table field.

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

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:

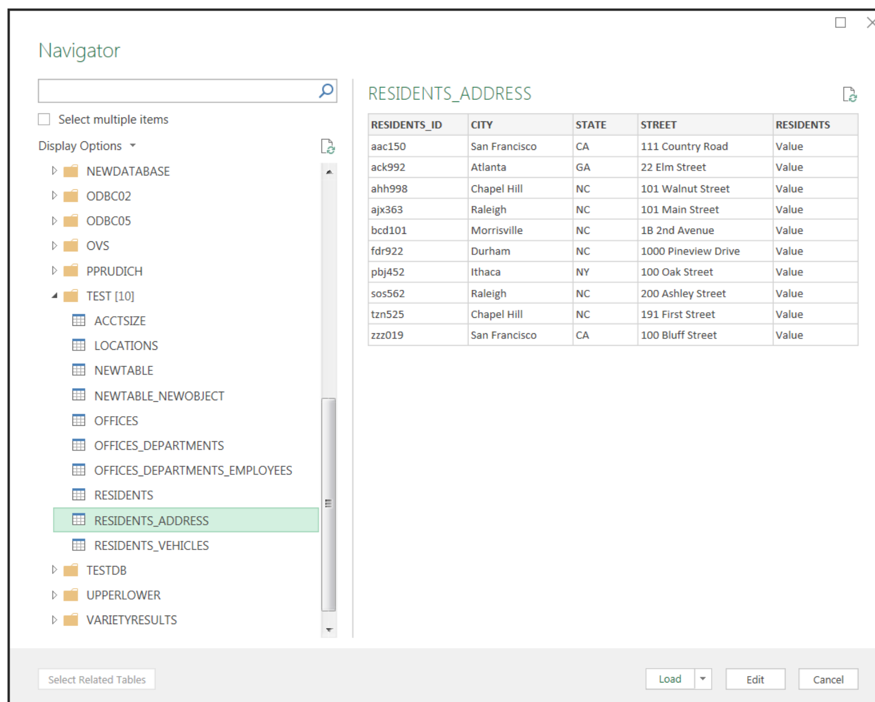
1. Open your workbook in Excel.
2. From the **Data** menu, select **Get Data**>**From Other Sources**>**From ODBC**.
3. The **From ODBC** dialog appears.

   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 options 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 options 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.

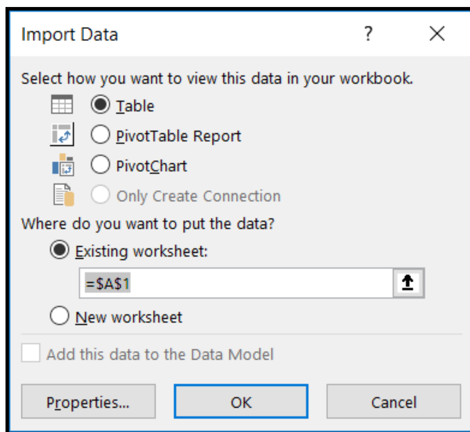


   From the list, select the tables you want to access. A preview of your data will appear in the pane on the right. Optionally, click **Edit** to modify the results using the Query Editor. Refer to the Microsoft Excel product documentation for detailed information on using the Query Editor.

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/.

3

Configuring and connecting to data sources

After you install the driver, you configure data sources to connect to the database. 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 all platforms, file data sources, which are simply text files, can be stored locally or on a network computer, and are accessible to other machines. 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 "Using a connection string" for an alphabetical list of driver connection string attributes and their initial default values.

For details, see the following topics:

- Environment settings
- Configuring data sources with the Configuration Manager
- Generating connection strings with the Configuration Manager
- Using a connection string
- Additional configuration methods for UNIX and Linux
- Testing connections and queries with the Configuration Manager
- Password Encryption Tool (UNIX/Linux only)
- Using a logon dialog box

- Authentication
- TLS/SSL encryption
- Proxy server
- MongoDB Atlas clusters
- Replica set failover for write operations
- Performance considerations

Environment settings

The first step in setting up and configuring the driver for use is to set environment settings and 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.

Windows environment variables

Before you can use your driver, you must set the PATH environment variable to include the path of the `jvm.dll` file of your Java™ Virtual Machine (JVM).

Note: During installation, the Windows installer sets the PATH environment variable to include the path of the JVM.

UNIX/Linux environment variables

The following topics guide you through setting the environment variables for UNIX/Linux platforms. You must set these environment variables before connecting with your driver.

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 HP-UX IPF, Linux, and Oracle Solaris

- `LIBPATH` on AIX

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

The product installer places a default system information file, named odbc.ini, that contains data sources in the product installation directory. See "Configuration through the system information (odbc.ini) file" 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.

See also

Configuration through the system information (odbc.ini) file on page 68

ODBCINST

The installer program places a default file, named odbcinst.ini, for use with DSN-less connections in the product installation directory. 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.

See also

DSN-less connections on page 71

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 environment, 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 driver is installed in /opt/odbc/lib, the following command attempts to load the 32-bit driver on Linux, where *xx* represents the version number of the driver:

```
ivtestlib/opt/odbc/lib/ivmongodbxx.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.

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

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:

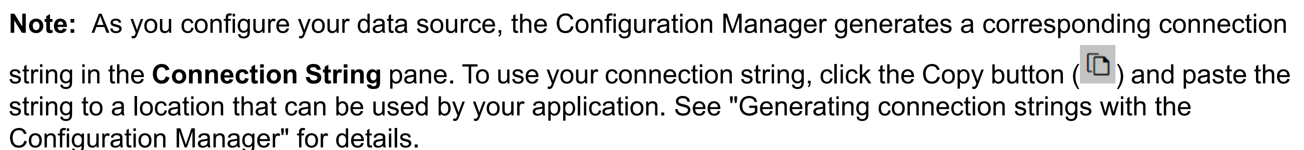
    ```
    rc = SQLSetEnvAttr(*henv, SQL_ATTR_APP_UNICODE_TYPE,
    (SQLPOINTER)SQL_DD_CP_UTF16, SQL_IS_INTEGER);
    ```

  - 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);
    ```

Configuring data sources with the Configuration Manager

The driver includes an enhanced setup dialog, the Progress DataDirect MongoDB Configuration Manager, that allows you to configure data sources, generate connection strings, test connections, and execute test queries. On Windows, data sources are stored in the Windows Registry. You can configure and modify data sources through the ODBC Administrator using the Configuration Manager, as described in this section.

Note: As you configure your data source, the Configuration Manager generates a corresponding connection string in the **Connection String** pane. To use your connection string, click the Copy button () and paste the string to a location that can be used by your application. See "Generating connection strings with the Configuration Manager" for details.

Note: Connection string attributes can be used to override the default values of the data source if you want to change these values at connection time.

To configure and test a data source:

1. Open the Windows ODBC Administrator.
2. Open the Configuration Manager through the **User DSN** or **System DSN** tab.
   - **User DSN**: If you are configuring an existing user data source, select the data source name and click **Configure** to display the Configuration Manager in your browser.

     If you are configuring a new user data source, click **Add** to display a list of installed drivers. Select your driver and click **Finish** to display the Configuration Manager.
   - **System DSN**: If you are configuring an existing system data source, select the data source name and click **Configure** to display the Configuration Manager in your browser.

     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 Configuration Manager.
   - **Note:** Configuring an existing or new file data source using the File DSN tab is not currently supported with the configuration manager.

   The MongoDB Configuration Manager window opens.
3. From the Configuration Manager window, provide values of the connection options you want to configure in the corresponding fields. To view more options, select the tabs at the top of the page. See "Connection option descriptions" for descriptions of the supported options.

   Note: See "Connection string examples" for a list of required options used for different configurations. The options and settings described in that section apply to all methods of configuration.
4. At any point during the process, you can click **Test Connect** to attempt to connect to the server with your settings. In the Test Connection window:

   a) Provide values for any fields required by your server. Note that the information you enter in the logon dialog box during a test connect is not saved.

   b) Optionally, in the **Test Query** field, enter any SQL queries you want to execute during the test. For example:

   ```
   SELECT * FROM INFORMATION_SCHEMA.SYSTEM_TABLES
   ```

   c) Click **Execute**.

   If successful, the window displays a confirmation message and, if a query was specified, the results of the query.
5. Click **Save** to apply your values as the default when connecting with the data source.

Generating connection strings with the Configuration Manager

Note: The Configuration Manager is currently supported only on Windows platforms.

The Progress DataDirect MongoDB Configuration Manager supports generating connection strings that can be used with your application. To generate a connection string, create a data source as described in "Configuring data sources with the Configuration Manager." As you provide connection option values, the Configuration Manager generates a connection string in the **Connection String** pane that corresponds to the data source.

In addition to providing values for connection option fields, you can manually edit your string by clicking the Edit button ( ). Note that editing your connection string also changes the values for the data source.

After you are done configuring the connection options, click **Test Connect** to test your connection string. See "Testing connections and queries with the Configuration Manager" for more information.

To use your string, click the Copy button ( ) and paste the string to a location that can be used by your application.

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 driver for UNIX, Linux or Windows is:

```
DSN=MongoDB;USER=JOHN;PWD=XYZZY
```

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

```
FILEDSN=MongoDB.dsn;USER=JSMITH;PWD=XYZZY
```

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

```
DRIVER=DataDirect 8.1 MongoDB;HostName=myserver;User=JSMITH;PWD=secret;
```

See also

Connection option descriptions on page 105
Connection string examples on page 22

Additional configuration methods for UNIX and Linux

This section contains configuration methods that are specific to the UNIX and Linux environments.

Configuration through the system information (odbc.ini) file

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.

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:

```
MongoDB=DataDirect 8.1 MongoDB
```

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, `[S4HANA]`. 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 the "Connection option descriptions" for details.

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*.

Sample default odbc.ini file

The following is a sample `odbc.ini` file that the installer program 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]
MONGODB=DataDirect 8.1 MongoDB

[MONGODB]
Driver=ODBCHOME/lib/ivmongodb81.so
Description=DataDirect 8.1 MongoDB Driver
ApplicationUsingThreads=1
ArrayNormalizationThreshold=12
AuthenticationMethod=0
AuthenticationDatabase=
BrokerIdleTimeout=600
BrokerPingInterval=30
BrokerPortNumber=1183
ColumnDiscoverySampleSize=1000
CreateMap=2
CryptoProtocolVersion=
DatabaseName=
EnableDNSLookup=1
EncryptionMethod=0
ExtendedOptions=
FetchSize=100
FlattenArrayBase=0
HostName=
HostNameInCertificate=
InitializationString=
JSONColumns=1
```

```
JVMArgs=-Xmx256m
JVMClasspath=
JVMPath=
KeyPassword=
Keystore=
KeystorePassword=
KeywordConflictSuffix=
LeadingUnderscoreReplacement=
LegacyVirtualKeys=0
LogConfigFile=ddlogging.properties
LoginTimeout=
MaxConnectionsPerServer=1024
MinVarcharSize=1
NetworkMessageCompressors=none
Password=
PortNumber=27017
ProxyHost=
ProxyPassword=
ProxyPort=0
ProxyUser=
QualifyNormalizedNames=1
ReadOnly=0
ReadPreference=1
ReplicaSetName=
ReportCodepageConversionErrors=0
SchemaFilter=
SchemaFormat=2
SchemaMap=
ServerIdleTimeout=300
ServerLaunchTimeout=30
ServerPortNumber=19968
SpecialCharBehavior=0
SQLEngineMode=2
StringTruncationMethodForWrites=1
TimestampFormat=1
TransactionMode=0
Truststore=
TruststorePassword=
UppercaseIdentifiers=1
User=
ValidateServerCertificate=
VarcharThreshold=4000

[ODBC]
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:**

  a) Using a text editor, open the `odbc.ini` file.

  Note: In addition to a text editor, you can also modify the `odbc.ini` file using the Configuration Manager. See "Configuring data sources with the Configuration Manager" for details.

  ---

  b) Modify the default values for attributes in the data source definitions as necessary based on your system specifics.

See "Connection option descriptions" for other specific attribute values.

c) 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:**

a) Using a text editor, open the `odbc.ini` file.

b) Copy an appropriate existing default data source definition and paste it to another location in the file.

c) 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, `[My Datasource]`.

d) Modify the attributes in the new definition as necessary based on your system specifics.

See "Connection option descriptions" for other specific attribute values.

e) 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.

f) 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 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.

See also

Configuring data sources with the Configuration Manager on page 65
Connection option descriptions on page 105

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.

Note: The driver and driver manager support ASCII and UTF-8 encoding in the `odbcinst.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 Drivers]` that lists installed drivers, for example,

```
DataDirect 8.1 MongoDB=Installed
```

This section also includes additional information for each driver.

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.

See also

ODBCINST on page 63

Configuration through the system information (odbc.ini) file on page 68

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 8.1 MongoDB=Installed

[DataDirect 8.1 MongoDB]
Driver=ODBCHOME/lib/ivmongodb81.so
JarFile=ODBCHOME/java/lib/mongodb.jar
APILevel=0
ConnectFunctions=YYY
CPTimeout=60
DriverODBCVer=3.52
FileUsage=0
SQLLevel=0
UsageCount=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.

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 "Configuring and connecting to data sources" for a general description of ODBC data sources on both Windows, UNIX, and Linux.

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 8.1 MongoDB
HostName=myserver
User=JSMITH
Password=secret
```

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\mongodb.dsn
```

or, on UNIX and Linux:

```
FILEDSN=/home/users/john/filedsn/mongodb.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 "Configuration through the system information (odbc.ini) file" 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/mongodb.dsn;User=john;PWD=test01
```

See also

Configuring and connecting to data sources on page 61
DSN-less connections on page 71
Configuration through the system information (odbc.ini) file on page 68

Testing connections and queries with the Configuration Manager

Note: The Configuration Manager is currently supported only on Windows platforms.

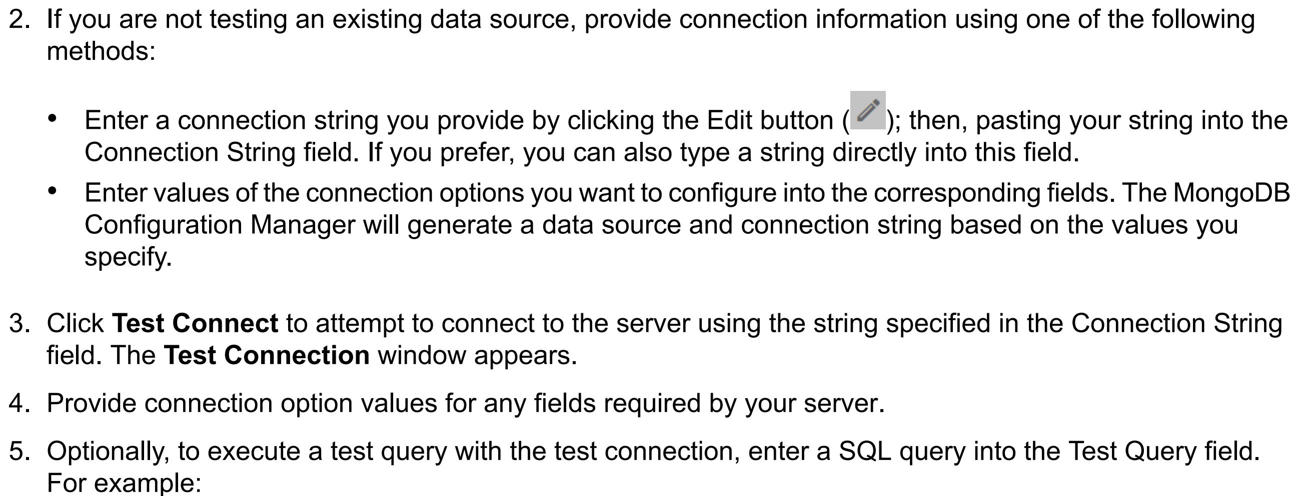
You can quickly test data sources, connection strings and queries using Progress DataDirect MongoDB Configuration Manager.

To test your connection and query:

1. Open the Windows ODBC Administrator. Then, select or add a data source to open the MongoDB Configuration Manager.

   For detailed information on launching the Configuration Manager, see "Configuring data sources with the Configuration Manager."

2. If you are not testing an existing data source, provide connection information using one of the following methods:

   - Enter a connection string you provide by clicking the Edit button (); then, pasting your string into the Connection String field. If you prefer, you can also type a string directly into this field.
   - Enter values of the connection options you want to configure into the corresponding fields. The MongoDB Configuration Manager will generate a data source and connection string based on the values you specify.

3. Click **Test Connect** to attempt to connect to the server using the string specified in the Connection String field. The **Test Connection** window appears.
4. Provide connection option values for any fields required by your server.
5. Optionally, to execute a test query with the test connection, enter a SQL query into the Test Query field. For example:

   ```
   SELECT * FROM INFORMATION_SCHEMA.SYSTEM_TABLES
   ```

6. Click **Execute**.

   If successful, the window displays a confirmation message and, if a query was specified, the results of the query.

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. To connect, provide the values described in the following sections; then, click **OK** to complete the logon.

In the dialog box, provide the following information:

- In the Host Name field, specify the name or the IP address of the server to which you want to connect. For example, `myserver`.
- In the User field, type the user name that is used to connect to the MongoDB server.
- In the Password, type the password used to connect to the MongoDB server.

Authentication

The driver supports the following authentication methods:

- *Basic authentication* authenticates using the specified user IDs and passwords.
- *LDAP authentication* authenticates using user ID and password information centrally maintained in LDAP.
- *Kerberos authentication* authenticates by using Kerberos authentication protocol.

By default, the driver is configured to use basic authentication (`AuthenticationMethod=0`).

User ID and password authentication

The driver supports user ID and password authentication.

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

- Set the Host Name (HostName) option to specify the name or the IP address of the MongoDB server to which you want to connect. For example, `myserver`.

- Set the User (User) option to specify your user ID.
- Set the Password (Password) option to specify the password for the user connecting to the instance.
- Optionally, set the Database Name (DatabaseName) option to specify the name of the database to which you are connecting.
- Optionally, set the Authentication Database (AuthenticationDatabase) option to specify the database in which your user ID was created. This value allows you to explicitly select a set of credentials and permissions when the same user ID was created in multiple databases on the server.

  Note: We recommend specifying a value for the Authentication Database to ensure the correct permissions are used for your connection. If you do not specify a value for this option, the driver attempts to use your user ID with the database specified by the Database Name (DatabaseName) option. If your user ID was not created in the specified database, the driver will attempt to connect to the Admin database using your user ID and password.

- Optionally, specify values for any additional options you want to configure.

Note: The User 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.

The following examples show the connection information required to establish a session using user ID and password authentication.

Connection string

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=mydb2;DatabaseName=mydb;
      HostName=myserver;User=jsmith;Password=secret;
```

odbc.ini

```
[MongoDB]
Driver=ODBCHOME/lib/ivmongodb81.so
...
Description=DataDirect 8.1 MongoDB
...
AuthenticationDatabase=mydb2
...
DatabaseName=mydb
...
HostName=myserver
...
User=jsmith
...
Password=secret
...
```

See also

Connection option descriptions on page 105

LDAP authentication

LDAP (Lightweight Directory Access Protocol) is a directory information service that allows you to centrally store information and share it across an IP network. In an LDAP service, information is stored in objects called entries, which can contain a variety of data—including authentication information. LDAP entries are often used to store authentication information because data storage is centralized, thereby simplifying maintenance when changes occur.

The driver supports user ID and password authentication.

To configure the driver to use LDAP authentication:

- Set the Host Name (HostName) option to specify the name or the IP address of the MongoDB server to which you want to connect. For example, `myserver`.
- Set the AuthenticationMethod to `17`.
- Set the User (User) option to specify your user ID.
- Set the Password (Password) option to specify the password for the user connecting to the instance.
- Optionally, set the Database Name (DatabaseName) option to specify the name of the database to which you are connecting.
- Optionally, specify values for any additional options you want to configure.

Note: The User 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.

The following examples show the connection information required to establish a session with LDAP authentication.

Connection string

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationMethod=17;DatabaseName=mydb;
       HostName=myserver;User=jsmith;Password=secret;
```

odbc.ini

```
[MongoDB]
Driver=ODBCHOME/lib/ivmongodb81.so
...
Description=DataDirect 8.1 MongoDB
...
AuthenticationMethod=17
...
DatabaseName=mydb
...
HostName=myserver
...
User=jsmith
...
Password=secret
...
```

See also

Connection option descriptions on page 105

Kerberos authentication

The driver supports the *Kerberos authentication*. 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 Windows Active Directory 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 Windows client, the application user does not explicitly need to obtain a TGT. Windows Active Directory automatically obtains a TGT for the user.

If the application uses Kerberos authentication from a UNIX or 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.

To configure the driver to use Kerberos authentication:

- Set the Authentication Method (AuthenticationMethod) option to `4`.
- Set the Host Name (HostName) option to specify the name or the IP address of the server to which you want to connect.
- Optionally, set the Database Name (DatabaseName) option to specify the name of the database to which you are connecting.
- Optionally, set the Port Number (PortNumber) option to specify the port number of the server listener. The default is `27017`.
- Optionally, set the Service Principal Name (ServicePrincipalName) option to specify the service principal name to be used by the driver.
- Set the User Name (LogonID) option to specify your user name.

  Note: The User option is not required to be stored in the connection string. It can also be sent separately by the application using the `SQLConnect` ODBC API. For `SQLDriverConnect` and `SQLBrowseConnect`, the option needs to be specified in the connection string.

- Optionally, specify values for any additional options you want to configure.

The following examples show the connection information required to establish a connection using Kerberos authentication.

Connection string

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=4;DatabaseName=mydb;
       HostName=myserver;PortNumber=27017;User=jsmith;
```

odbc.ini

```
[MongoDB]
Driver=ODBCHOME/lib/ivmongodb81.so
...
Description=DataDirect 8.1 MongoDB
...
AuthenticationDatabase=4
...
DatabaseName=mydb
...
HostName=myserver
...
PortNumber=27017
...
User=jsmith
...
```

See also

Connection option descriptions on page 105

TLS/SSL encryption

The driver supports TLS/SSL data encryption. TLS/SSL works by allowing the client and server to send each other encrypted data that only they can decrypt. 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.

Note: The version of TLS/SSL that is used and which cryptographic algorithm is used depends on which JVM you are using. Refer to your JVM documentation for more information about its TLS/SSL support.

TLS/SSL server authentication

When the client makes a connection request, the server presents its public certificate for the client to accept or deny. The client checks the issuer of the certificate against a list of trusted Certificate Authorities (CAs) that resides in an encrypted file on the client known as a *truststore*. If the certificate matches a trusted CA in the truststore, an encrypted connection is established between the client and server. If the certificate does not match, the connection fails and the driver generates an error.

Most truststores are password-protected. The driver must be able to locate the truststore and unlock the truststore with the appropriate password. Two connection options are available to the driver to provide this information: Trust Store (Truststore) and Trust Store Password (TruststorePassword). The value of Trust Store is a pathname that specifies the location of the truststore file. The value of Trust Store Password is the password required to access the contents of the truststore.

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.

Finally, the connection option, Host Name In Certificate (HostNameInCertificate), allows an additional method of server verification. When a value is specified for Host Name In Certificate, it must match the common name of the host in the Subject of the certificate. This prevents malicious intervention between the client and the server and ensures that the driver is connecting to the server that was requested.

The following examples show how to configure the driver to use data encryption via the SSL server authentication. In this configuration, since `ValidateServerCertificate=1`, the driver validates the certificate sent by the server and the host name specified by the HostNameInCertificate option.

Using a connection string:

```
DRIVER=DataDirect 8.1 MongoDB;DatabaseName=mydb;EncryptionMethod=1;HostName=myserver;
  HostNameInCertificate=MySubjectAltName;KeyPassword=key_password;
  ValidateServerCertificate=1;User=jsmith;Password=secret;
```

Using an `odbc.ini` file:

```
Driver=ODBCHOME/lib/ivmongodbxx.so
Description=DataDirect MongoDB
...
DatabaseName=mydb
...
EncryptionMethod=1
...
HostName=myserver
HostNameInCertificate=MySubjectAltName
...
Truststore=TrustStoreName
TruststorePassword=TSXYZZY
...
ValidateServerCertificate=1
...
```

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.

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

- Specify values for the Java system properties javax.net.ssl.keyStore, javax.net.ssl.keyStorePassword, and javax.net.ssl.keyPassword using the JVM Arguments (JVMArgs) connection option, which sets these properties for the Java based SQL engine component of the driver. For example:

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

- Specify values for the Key Store and Key Store Password connection options. See the following examples for demonstrations of configuring data encryption in a connection string and the `odbc.ini` file..

The following examples show how to configure the driver to use data encryption via the SSL client authentication. In this configuration, since `ValidateServerCertificate=1`, the driver validates the certificate sent by the server and the host name specified by HostNameInCertificate.

Using a connection string:

```
DRIVER=DataDirect 8.1 MongoDB;DatabaseName=mydb;EncryptionMethod=1;
  HostName=myserver;HostNameInCertificate=MySubjectAltName;Keystore=KeyStoreName;
  KeystorePassword=YourKSPassword;ValidateServerCertificate=1;User=jsmith;Password=secret;
```

Using the `odbc.ini` file:

```
Driver=ODBCHOME/lib/ivmongodbxx.so
Description=DataDirect MongoDB
...
DatabaseName=mydb
...
EncryptionMethod=1
...
HostName=myserver
HostNameInCertificate=MySubjectAltName
...
Keystore=KeyStoreName
KeystorePassword=YourKSPassword
...
ValidateServerCertificate=1
...
```

Supported keystores and truststores

The driver's TLS/SSL communication layer is implemented through its SQL Engine component. Because the SQL Engine is Java based, the keystore/truststore files used for TLS/SSL encryption must be in the following Java compatible formats:

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

Proxy server

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

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

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

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

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

Alternatively, you can use the Proxy Host (`ProxyHost`), Proxy Port (`ProxyPort`), Proxy User (`ProxyUser`), and Proxy Password (`ProxyPassword`) connection options. See "Connection Option Descriptions" for details about these attributes.

See also

Connection option descriptions on page 105

MongoDB Atlas clusters

MongoDB Atlas is a hosted solution for accessing your MongoDB data in the cloud. Because MongoDB Atlas is a clustered implementation, you configure the driver to connect through the domain, instead of directly to a server. At connection, the driver performs a DNS lookup to discover the member nodes, then connects to an available node in the cluster.

Note: In addition to MongoDB Atlas, you can connect to other clustered environments by specifying the domain name using the Host Name (HostName) option. However, other settings for that data source, such as authentication, are likely to differ from those described in this topic. Note that the Enable DNS Lookup (EnableDNSLookup) option must be set to `1` (the default) to connect to a clustered environment.

To connect to a MongoDB Atlas cluster:

- Set Host Name (HostName) to specify the domain name of your MongoDB Atlas cluster to which you want to connect. For example, `myaltas.mongodb.net`.
- Optionally, set Database Name (DatabaseName) to specify the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries.
- Set PortNumber (PortNumber) to specify the port number of the server listener. The default is `27017`.
- Set Authentication Method (AuthenticationMethod) to `0` (the default).
- Set Encryption Method (EncryptionMethod) to `1` (SSL).
- Set User (User) to specify the user name that is used to connect to the MongoDB database. For example, `jsmith`.
- Set Password (Password) to specify the password used to connect to your MongoDB database.
- Optionally, set the Authentication Database (AuthenticationDatabase) option to specify the database in which your user ID was created. This value allows you to explicitly select a set of credentials and permissions when the same user ID was created in multiple databases on the server.

Note: We recommend specifying a value for the Authentication Database to ensure the correct permissions are used for your connection. If you do not specify a value for this option, the driver attempts to use your user ID with the database specified by the Database Name option. If your user ID was not created in the specified database, the driver will attempt to connect to the Admin database using your user ID and password.

The following examples include the option attributes required for connecting to a MongoDB Atlas cluster.

Connection string

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=mydb2;DatabaseName=mydb;
  EncryptionMethod=1;HostName=//myaltas.mongodb.net;PortNumber=27017;User=jsmith;
  Password=secret;
```

odbc.ini

```
[MongoDB]
Driver=ODBCHOME/lib/ivmongodb81.so
...
Description=DataDirect 8.1 MongoDB
...
AuthenticationDatabase=mydb2
...
DatabaseName=mydb
...
EncryptionMethod=1
...
HostName=//myaltas.mongodb.net
...
PortNumber=27017
...
User=jsmith
...
Password=secret
...
```

See also

Connection option descriptions on page 105

Replica set failover for write operations

Write operations for replica sets are performed exclusively through connections to the primary member. However, sometimes the primary member can become unavailable, for example, due to hardware failure or traffic overload. If this occurs, the other members elect a secondary member to assume the role of the primary. The ability to promote the secondary member during outages ensures uninterrupted availability of write operations and access to the most current version of data.

When replicate set failover is enabled, the driver handles this behavior by attempting to connect to the primary member for each write operation. If the connection fails, the driver repeats the discovery process until it finds the new primary member or the maximum number of retries have been attempted. You can enable replica set failover for write operations by specifying the name of your replica set using the Replica Set Name (ReplicaSetName) option.

Note: When a value is specified for Replica Set Name, write operations will not fail when connected to secondary member for read operations. The driver will always attempt to connect to the primary member for write operations.

To configure replica set name failover for write operations:

- Set Host Name (HostName) to specify the name or the IP address of the MongoDB server to which you want to connect. For example, `mymongodbserver`.
- Set Database Name (DatabaseName) to specify the name of the database to which you want to connect. This value is used as the default qualifier for unqualified table names in SQL queries. Required for User/ID password authentication.
- Set Port Number (PortNumber) to specify the port number of the server listener. The default is `27017`.
- Set Replica Set Name (ReplicaSetName) to specify the name of the replica set against which you want to execute write operations.

The following examples include the options required for connecting with no authentication, replica set failover for write operations enabled, and replica set failover for read operations set to primary preferred .

Connection string

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationMethod=15;DatabaseName=mydb;
  HostName=myserver;PortNumber=27017;ReplicaSetName=MyReplicaSet;
```

odbc.ini

```
[MongoDB]
Driver=ODBCHOME/lib/ivmongodb81.so
...
Description=DataDirect 8.1 MongoDB
...
AuthenticationMethod=15
...
DatabaseName=mydb
...
HostName=myserver
...
ReplicaSetName=MyReplicaSet
...
```

See also

Connection option descriptions on page 105

Performance considerations

Application Using Threads (ApplicationUsingThreads): The driver coordinates concurrent database operations (operations from different threads) by acquiring locks. Although locking prevents errors in the driver, it also decreases performance. If your application does not make ODBC calls from different threads, the driver has no reason to coordinate operations. In this case, the ApplicationUsingThreads attribute should be disabled (set to `0`).

Note: If you are using a multi-threaded application, you must enable the Application Using Threads option.

Enable DNS Lookup (EnableDNSLookup): Enable DNS Lookup specifies whether the driver performs a DNS lookup to discover member nodes of a cluster when attempting to connect. If you are connecting to a non-clustered environment, you can bypass the lookup and improve connection time by disabling this option (`EnableDNSLookup=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.

Fetch Size (FetchSize): Reducing the number of round trips on the network to the approximate number of rows being fetched increases performance. For example, if your application normally fetches 200 rows, it is more efficient for the driver to fetch 200 rows at one time over the network than to fetch 50 rows at a time during four round trips over the network.

JVM Arguments (JVMArgs): This connection option can be used to address memory and performance concerns by adjusting the max Java heap size. By increasing the max Java heap size, you increase the amount of data the driver accumulates in memory. This can reduce the likelihood of out-of-memory errors and improve performance by ensuring that result sets fit easily within the JVM's free heap space.

Read Preference (ReadPreference): This connection option allows you to specify your preference for which replica set members are read when executing queries. Executing queries against primary members (read-write server nodes) returns the most recent version of the data, but increases the workload of the primary members and may negatively affect performance. If your application does not require the most recent version of data, consider setting this connection option to read from secondary members (read-only server nodes) to improve performance.

Result Memory Size (ResultMemorySize): Since Result Memory Size specifies the maximum size of an intermediate result set that the driver holds in memory, it can affect performance in two ways. First, if the value of the result set is larger than the value specified for Result Memory Size, the driver writes a portion of the result set to disk. Since writing to disk is an expensive operation, performance losses will be incurred. Second, when you remove any limit on the size of an intermediate result set by setting Result Memory Size to 0, you can realize performance gains for result sets that easily fit within the JVM's free heap space. However, the same setting can diminish performance for result sets that only just fit within the JVM's free heap space.

Schema Filter (SchemaFilter): Schema Filter provides a method to limit the database and collection pairs for which the driver fetches metadata. If your application only needs to access a subset of the collections you have access to, you can significantly improve connection times by providing a value for this option.

See also

Connection option descriptions on page 105

4

Using the SQL engine server

Some applications may experience problems loading the JVM required for the SQL engine because the process exceeds the available heap space. If your application experiences problems loading the JVM, you can configure the driver to operate with the SQL engine in a separate process with its own JVM.

The driver supports the following SQL engine modes:

- **Broker mode**: To efficiently manage memory and resources, the Broker automatically starts and stops itself based on configuration settings. The Broker launches the driver's in memory SQL engine service when a connection to the database requires the SQL Engine to process SQL requests and return results. This mode eliminates the need for the application to explicitly start and stop the service.

  For details, see "Configuring the SQL engine server using the Configuration Manager" (Windows) or "Configuring Broker mode for the SQL engine server on UNIX/Linux."

- **Server mode**: The driver's SQL Engine runs in its own JVM process instead of loading the JVM into the same process as the application. Server mode requires the SQL engine server to be manually started and stopped.

  For details, see "Configuring the SQL engine server using the Configuration Manager" (Windows) or "Configuring server mode using Java options" (UNIX/Linux).

- **Direct mode**: The driver operates with the SQL Engine running as a sub-process to the application process that loaded the ODBC driver library.

  For details, see "Configuring the SQL engine server using the Configuration Manager" (Windows).

- **Auto mode**: The driver attempts to run in server mode; however, if the SQL engine server is unavailable, the SQL engine will failover to run in direct mode.

  For details, see "Configuring the SQL engine server using the Configuration Manager" (Windows) or "Configuring server mode using Java options" (UNIX/Linux).

By default:

- **Windows**: The driver operates in server mode by default.
- **UNIX/Linux**: The driver operates in direct mode by default.

Note: You must be an administrator to start or stop the service, or to configure any settings for the service. Alternatively, you can run the driver in Broker mode to remove this dependency on the external SQL engine service and allow for the service to be managed automatically and on-demand.

See the following sections for details on configuring the SQL engine server on your platform.

For details, see the following topics:

- Configuring the SQL engine server using the Configuration Manager
- Configuring Broker mode for the SQL engine server on UNIX/Linux
- Configuring server mode using Java options
- Configuring Java logging for the SQL engine server

Configuring the SQL engine server using the Configuration Manager

Before you begin: If you are using server mode, you must start the SQL engine server before connecting. Before starting the SQL engine server, choose a directory to store the local database files using the Schema Map (SchemaMap) option. Make sure that you have the correct permissions to write to this directory.

Note: The Configuration Manager is currently supported only on Windows platforms. To configure the SQL engine on UNIX/Linux platforms, see "Configuring Broker mode for the SQL engine server on UNIX/Linux" or "Configuring the SQL engine server using Java options."

The following sections describe how to configure and start the SQL engine server using the Configuration Manager.

1. Open your data source using the Configuration Manager; then, select the **SQL Engine** tab.
2. Set the SQL Engine Mode (SQLEngineMode) connection option to one of the following values:
   - **0 - Auto**: The SQL engine attempts to run in server mode first, but will failover to direct mode if server mode is unavailable.
   - **1 - Server**: The SQL engine runs exclusively in server mode.
   - **3 - Broker**: The SQL engine runs exclusively in Broker mode.

   Note: By default, SQL Engine Mode is set to **1 - Server** for Windows and **2 - Direct** for UNIX/Linux.

   The fields associated with server mode will become exposed, and the **Start** button appears.
3. Provide values for the following options:

   For server mode only:

- **Server Port Number**: Specifies a valid port on which the SQL engine listens for requests from the driver. The default value depends on your platform:

  32-bit: `19968`

  64-bit: `19967`

For Broker mode only:

- **Broker Port Number**: Specifies a valid port on which the Broker listens for client connection requests. The default value is `1183`.
- **Max Connections Per Server**: Specifies the number of connections that can be granted to a server before another server is created. The default value is `1024` connections.
- **Server Launch Timeout**: Specifies the number of seconds the Broker waits for a server process to initialize before determining that is not available. The default value is `30` seconds.
- **Server Idle Timeout**: Specifies the number of seconds the SQL engine server can be without active connections before the Broker issues a `SHUTDOWN` command to stop the server. The default value is `300` seconds.
- **Broker Idle Timeout**: Specifies the number of seconds the Broker continues to run after all the SQL engine servers have been stopped (shutdown). This option allows for the Broker to shut itself down while there are no active connections to the database. When a new connection request is made by the application, the Broker restarts itself and spawns a SQL engine process to serve the database requests made through the driver. The default value is `600` seconds.
- **Broker Ping Interval**: Specifies the number of seconds between status heartbeats from a SQL engine server process to the Broker. The default value is `30` seconds.

For server and Broker mode:

- **JVM Path**: Specifies fully qualified path to the Java SE 8 or higher JVM executable that you want to use to run the SQL engine server. The path must not contain double quotation marks. The default value is:

  install_dir`\jre\bin\java.exe`

- **JVM Arguments**: A string that contains the arguments that are passed to the JVM that the driver is starting. The location of the JVM must be specified on your PATH. See "JVM Arguments" for details. The following are the default values:
  - For the 32-bit driver in direct or Broker mode: `-Xmx256m`
  - For all other configurations: `-Xmx1024m`
- **JVM Classpath**: Specifies the CLASSPATH for the JVM used by the driver. See "JVM Classpath" for details.

Note: After the initial configuration, in order for changes to the required and optional connection option values to take effect, you must restart the SQL engine server.

---

4. Click **Save** to save your changes
5. Start the SQL engine service:
   - Auto or server mode: Click **Start** to run the SQL engine service A message is displayed that indicates whether the SQL engine was able to successfully run.

- Broker mode: The SQL engine service automatically launches when you establish a connection.

You have configured and started your SQL engine server.

See also

Configuring server mode using Java options on page 92
JVM Classpath on page 129
JVM Arguments on page 128

Stopping the SQL engine server using the Configuration Manager

The following describes how to stop the SQL engine server using the Configuration Manager. This content applies only if your driver is configured to use server mode (`SQLEngineMode=1`).

Note that If you are using direct mode, then the SQL Engine process ends when the application process ends. However, if you are using broker mode, then the SQL Engine process ends based on the setting of the Broker Idle Time (BrokerIdleTimeout) option.

To stop the SQL engine server:

1. Open the Configuration Manager and select the SQL Engine tab.
2. From the SQL Engine Mode drop-down list, select **0 - Auto** or **1 - Server**.
3. Click **Stop** to stop the service. A message is displayed to confirm that the service stopped.
4. Click **Exit** to close the Configuration Manager.

Configuring Broker mode for the SQL engine server on UNIX/Linux

When Broker mode is enabled, the SQL engine operates in an external Java process that is monitored by the driver's Broker. To efficiently provision memory and resources, the Broker automatically starts and stops the service as needed. This setting also eliminates the need for the application or users to explicitly start and stop the service.

Note that, unlike server mode, you do not need to launch the SQL engine server before establishing a connection. The Broker will launch the SQL engine server automatically when attempting to connect.

To configure the driver to use Broker mode:

- Configure the minimum options required for a connection:
  - Set the Host Name (HostName) option to specify the name or the IP address of the MongoDB server to which you want to connect. For example, `myserver`.
  - Set the Schema Map (SchemaMap) option to specify the fully qualified path of the configuration file where the relational map of native data is written.
  - Set the Port Number (PortNumber) option to specifies the port number of the server listener. The default value is `27017`.
  - Set options required for authentication to your server. See "Authentication" for a description of authentication methods and their requirements.

- Set the Database Name (DatabaseName) option to specify the name of the database to which you are connecting.

- Set the SQL Engine Mode (SQLEngineMode) option to `3` (Broker).
- Optionally, set the Broker Port Number (BrokerPortNumber) option to specify a valid port on which the Broker listens for client connection requests. The default value is `1183`.
- Optionally, set the Max Connections Per Server (MaxConnectionsPerServer) option to specify the number of connections that can be granted to a server before another server is created. The default value is `1024`.
- Optionally, set the Server Launch Timeout (ServerLaunchTimeout) option to specify the number of seconds the Broker waits for a server process to initialize before determining that is not available. The default value is `30` seconds.
- Optionally, set the Server Idle Timeout (ServerIdleTimeout) option to specify the number of seconds the SQL engine server can be without active connections before the Broker issues a `SHUTDOWN` command to stop the server. The default value is `300` seconds.
- Optionally, set the Broker Idle Timeout (BrokerIdleTimeout) option to specify the number of seconds the Broker continues to run after all the SQL engine servers have been stopped (shutdown). This option allows you to shut down an idle Broker automatically to allow for additional connection requests from client processes. The default value is `600` seconds.
- Optionally, set the Broker Ping Interval (BrokerPingInterval) option to specify the number of seconds between status heartbeats from a SQL engine server process to the Broker. The default value is `30` seconds.
- Optionally, set the JVM Path (JVMPath) option to specify the fully qualified path to the Java SE 8 or higher JVM executable that you want to use to run the SQL engine server. The path must not contain double quotation marks.
- Optionally, set the JVM Arguments (JVMArgs) option to specify a string that contains the arguments that are passed to the JVM that the driver is starting. The location of the JVM must be specified on your PATH. See "JVM Arguments" for details. The following are the default values:
  - For the 32-bit driver in direct or Broker mode: `-Xmx256m`
  - For all other configurations: `-Xmx1024m`
- Optionally, set the JVM Classpath (JVMClasspath) option to specify the CLASSPATH for the JVM used by the driver. See "JVM Classpath" for details.

The following examples demonstrate the minimum configuration required to enable Broker mode using user ID and password authentication. The Broker mode is enabled, but the connection uses the default settings for the rest of the Broker-related options.

Connection string

```
DRIVER=DataDirect 8.1 MongoDB;AuthenticationDatabase=admin;DatabaseName=mydb;
      HostName=myserver;PortNumber=27017;SQLEngineMode=3;User=jsmith;
      Password=secret;
```

odbc.ini

```
[MongoDB]
Driver=ODBCHOME/lib/ivmongodb81.so
...
Description=DataDirect 8.1 MongoDB
...
AuthenticationDatabase=admin
...
DatabaseName=mydb
```

```
...
HostName=myserver
...
PortNumber=27017
...
SQLEngineMode=3
...
Password=secret
...
User=jsmith
...
```

See also

Authentication on page 75
Connection option descriptions on page 105

Configuring server mode using Java options

Before you start: If you are using server mode, you must start the SQL engine server before using the driver. Before starting the SQL engine server, verify that you have the correct permissions to write to the directory specified by the Schema Map (SchemaMap) option.

The following sections describe how to configure, start, and stop the SQL engine server using server mode on UNIX and Linux platforms.

Note: By default, SQL Engine Mode is set to `1` (Server) for Windows and `2` (Direct) for UNIX/Linux.

The SQL Engine runs in a java process and can be launched using a java command with the following format. See the "SQL engine server Java options" table for a description of these options.

```
java -Xmx<heap_size>m -cp "<jvm_classpath>" com.ddtek.jdbc.mongodb.phoenix.sql.Server
     -port <port_number> -Dhttp.proxyHost=<proxy_host> -Dhttp.proxyPort=<proxy_port>
     -Dhttp.proxyUser=<proxy_user> -Dhttp.proxyPassword=<proxy_password>
```

For example:

```
java -Xmx1024m -cp "/opt/Progress/DataDirect/ODBC_81_64bit/java/lib/mongodb.jar"
     com.ddtek.jdbc.mongodb.phoenix.sql.Server -port 19967
     -Dhttp.proxyHost=myhost@mydomain.com -Dhttp.proxyPort=12345
     -Dhttp.proxyUser=JohnQPublic -Dhttp.proxyPassword=secret
```

A confirmation message is returned once the server is online.

Note: After the initial configuration, in order for changes to these connection option values to take effect, you must restart the SQL engine server.

Table 26: SQL engine server Java options

Java Option	Description
Required Java Options	

Java Option	Description
`-cp`	Specifies the CLASSPATH for the Java Virtual Machine (JVM) used by the driver. The CLASSPATH is the search string the JVM uses to locate the Java jar files the driver needs. The driver's JVM is located on the following path: *install_dir*`/java/lib/mongodb.jar`
`-port`	Specifies a valid port on which the SQL engine listens for requests from the driver. We recommend specifying one of the following values: • `19968` (32-bit drivers) • `19967` (64-bit drivers)
Optional Java Options	
`-Xmx`	Specifies the maximum memory heap size, in megabytes, for the JVM. The default size is determined by your JVM. We recommend specifying a size no smaller than `1024`. **Note:** Although this option is not required to start the SQL engine server, we highly recommend specifying a value.
`-Dhttp.proxyHost`	Specifies the host name of the proxy server. The value specified can be a host name, a fully qualified domain name, or an IPv4 or IPv6 address.
`-Dhttp.proxyPort`	Specifies the port number where the proxy server is listening for HTTP and/or HTTPS requests.
`-Dhttp.proxyUser`	Specifies the user name needed to connect to the proxy server.
`-Dhttp.proxyPassword`	Specifies the password needed to connect to the proxy server.

Stopping the SQL engine server

The following describes how to stop the SQL engine server . This section applies only if your driver is configured to use server mode (`SQLEngineMode=1`).

Note: If you are using Broker mode, you do not need to stop the SQL engine server as described in this section. The SQL engine server will stop automatically when inactive for the period of time specified by the Server Idle Timeout (ServerIdleTimeout) option.

To stop the SQL engine server, choose one of the following:

- Using an application, execute `SHUTDOWN SQL`.
- From a command line, press `Ctrl` + `C`.

A message is returned to confirm that the service is stopped.

Configuring Java logging for the SQL engine server

Java logging for the SQL engine server can be configured using either the JVM or the driver.

For details, refer to "Configuring logging" in the *Progress DataDirect for ODBC Drivers Reference*.

5

Additional features and functionality

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

For details, see the following topics:

- MongoDB sharding support
- Using identifiers
- Parameter metadata support
- Binding parameter markers
- MongoDB views
- Local views
- MinKey and MaxKey values
- Packet logging

MongoDB sharding support

MongoDB employs sharding as a horizontal scaling solution for supporting large sets of data. It is designed to provide scalable throughput and storage capacity. To accomplish this, MongoDB shares logical databases across multiple independent replica sets (clustered servers), or shards. By distributing data across servers, operations are delegated only to the servers that store data relevant to the task. This increases the availability of servers and CPU capacity, resulting in increased throughput. If operations exceed the available processing capacity of the clusters, additional servers can be added to the cluster, which reduces the number of operations performed by each server and can improve performance. A similar principle applies to storage capacity, where servers can be added to accommodate increased storage requirements.

Caution: Although the driver connects to a MongoDB sharded cluster transparently, it is critical that the primary key column have unique values for every row (or document) in the table. The values for the default primary key of `_ID` are generated by a MongoDB database; however, in a sharded cluster, these values are not guaranteed to be unique across shards unless specifically configured in the MongoDB cluster. If duplicate identifiers are mapped to a relational view, write operations can produce undesired results.

Using identifiers

Identifiers are used to refer to objects exposed by the driver, such as tables and columns. The driver supports both quoted and unquoted identifiers for naming objects. The maximum length of both quoted and unquoted identifiers is 128 characters. Quoted identifiers must be enclosed in double quotation marks (""). A quoted identifier can contain any Unicode character, including the space character, and is case-sensitive. The driver recognizes the Unicode escape sequence \uxxxx as a Unicode character. You can specify a double quotation mark in a quoted identifier by escaping it with a double quotation mark. Unquoted identifiers must start with an ASCII alpha character and can be followed by zero or more ASCII alpha or numeric characters.

By default, the driver exposes native objects as unquoted, uppercase identifiers when creating the relational schema of your native data. Since native objects are case sensitive in MongoDB, the driver avoids naming conflicts by appending identifiers which have the same name but different cases with an underscore separator and integer (for example, `_1`). If one of the conflicting names contains only uppercase characters, that name will remain unaltered. For example, if the collections `Test`, `TEST`, and `test` are found, the driver will expose the collections as tables in the following manner:

Table 27: Name Conflict Resolution Example

Collection Name	Table Name
`Test`	`TEST_1`
`TEST`	`TEST`
`test`	`TEST_2`

The driver allows you to change default behavior related to identifiers and manage identifiers directly through the use of the connection options described in the following sections.

Uppercase Identifiers (UppercaseIdentifiers) option

The Uppercase Identifiers connection option determines whether native objects are mapped as unquoted, uppercase identifiers (the default) or quoted, mixed case identifiers that correspond directly with native object names. Therefore, as an alternative to the driver's default behavior, you can use Uppercase Identifiers to retain the names of native objects in the relational view of your data. When Uppercase Identifiers is set to `false`, the driver maps the names of native objects as quoted identifiers, maintaining the case of native object names in the relational view of native data. If these identifiers are called in a SQL statement, the statement must enclose the identifiers in double quotation marks and they must exactly match the case of the identifier name. For example, when `UppercaseIdentifiers=false`, you would use the following statement to query the Account table:

```
SELECT "id", "name" FROM "Account"
```

The setting for Uppercase Identifiers also affects the use of catalog functions. When object names are passed as arguments to catalog functions, the case of the value must match the case of the name in the database. If `Uppercase Identifiers=true` (the default) when the schema map was created, the value passed to the catalog function must be uppercase because unquoted identifiers are automatically converted to uppercase by the driver. If `UppercaseIdentifiers=false` when the schema definition was created, the value passed to the catalog function must match the case of the name as it was defined. In addition, when `UppercaseIdentifiers=false`, object names in results returned from catalog functions are returned in the case that they are stored in the database.

Keyword Conflict Suffix (KeywordConflictSuffix) option

You can use the Keyword Conflict Suffix connection option to avoid naming conflicts when the name of an object corresponds to the name of a SQL engine keyword. Keyword Conflict Suffix specifies a string of up to five alphanumeric characters that the driver appends to any object or field name that conflicts with a SQL engine keyword. For example, if you specify `KeywordConflictSuffix=TAB`, the driver maps the Case object to CASETAB.

Leading Underscore Replacement (LeadingUnderscoreReplacement) option

The Leading Underscore Replacement connection option allows you to replace leading underscores with a string when leading underscores are used in identifiers for collections and fields. For example, MongoDB collections automatically include the _id field. By specifying `LeadingUnderscoreReplacement=XX`, the _id field becomes the `XXID` column in the relational view of the data. In addition, any other fields or collections with a leading underscore would be modified in the same manner.

Qualify Normalized Name (QualifyNormalizedName) option

The Qualify Normalized Name option provides you with a method generate table names with greater context for collections that contain similar substructures. If set to `1` (Table) or `2` (FullPath), the driver prepends the table name with parent objects. Using these settings reduce the likelihood that naming conflicts would occur and potentially provides better context to child tables should there be similarly named nested objects in your collections.

Special Char Behavior (SpecialCharBehavior) option

The Special Char Behavior option allows you to determine how the driver handles the mapping of native identifiers containing characters that would require them to be quoted in SQL statements. This option provides a method to choose to continue using identifiers that require quotation marks or for the driver to modify affected identifier names so that quotation marks are not required. By default, the driver removes any characters that are not part of a legal, unquoted SQL identifier. In practice, this removes any characters that are not letters, digits, or underscores. For example, if the native name were `Cost of Customer Acquisition`, the mapped name would be `CostofCustomerAcquisition`. The default behavior eliminates the need to quote these identifiers, but changes the identifier's name when mapping it to the relational view.

Flatten Array Base option

When arrays are flattened into columns, the driver appends an underscore and the ordinal position to the column name (`<array_name>_<oridinal_location>`). You can specify the starting ordinal value, either a `0` or `1`, using the Flatten Array Base (FlattenArrayBase) option. For example, if you specified a value of 0, the first three names of columns of an array named vehicles would be `VEHICLES_0`, `VEHICLE_1`, `VEHICLE_0`. The default starting ordinal value is `1`.

JSON Columns option

The JSON Columns (JSONColumns) option determines whether the driver exposes complex columns as JSON values in addition to their normalized mapping. Exposing complex columns as JSON values can make certain operations more convenient, such as when tools use this data as a single object for communication. In rare instances where variations in data structure might cause this data to not be sampled, enabling this option also provides a method in which it can still be read as a JSON value. When enabled (`JSONColumns=1`), the driver will also expose complex columns values as JSON values, which can be returned if needed. When this option is disable (`JSONColumns=0`), the driver will only map complex values according to the standard normalization rules.

See also

Connection option descriptions on page 105

Parameter metadata support

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

Insert and Update statements

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

- `INSERT INTO FOO VALUES(?, ?, ?)`
- `INSERT INTO FOO (COL1, COL2, COL3) VALUES(?, ?, ?)`
- `UPDATE FOO SET COL1=?, COL2=?, COL3=? WHERE COL1 *operator* ? [{AND | OR} COL2 *operator* ?]`

where:

operator

is any of the following SQL operators:

`=`, `<`, `>`, `<=`, `>=`, and `<>`

Select statements

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

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

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

  ```
  SELECT * FROM FOO WHERE BAR > ?
  ```

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

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

  ```
  SELECT * FROM FOO WHERE (SELECT X FROM Y    WHERE Z = 1) < ?
  ```

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

```
SELECT COL1, COL2 FROM FOO WHERE COL1 = ? AND COL2 > ?
SELECT ... WHERE COLNAME = (SELECT COL2 FROM T2 WHERE COL3 = ?)
SELECT ... WHERE COLNAME LIKE ?
SELECT ... WHERE COLNAME BETWEEN ? AND ?
SELECT ... WHERE COLNAME IN (?, ?, ?)
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE COL1 < ?)
```

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

```
SELECT * FROM T1 WHERE COL = ? ORDER BY 1
```

Joins are supported. For example:

```
SELECT * FROM T1,T2 WHERE T1.COL1 = ?
```

Fully qualified names and aliases are supported. For example:

```
SELECT A, B, C, D FROM T1 AS A, T2 AS B WHERE A.A = ? AND B.B = ?
```

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.

MongoDB views

MongoDB databases allow you to create read-only views from existing collections or other views. MongoDB views are queryable objects that are typically used to join collections, add computed fields to collections, or exclude sensitive information. Similar to collection discovery, the driver detects MongoDB views when sampling and mapping data from the server. Data from the view is mapped to relational views using the same method as the rest of your collections (normalized, flattened, or mixed).

Be aware of the following behaviors when using MongoDB views with the driver:

- Relational tables mapped from MongoDB views are read-only
- Nested objects in a view are normalized to child tables in accordance to the driver's relational mapping behavior (normalized, flattened, or mixed). This is the same behavior used when mapping collections to the relational view.

Local views

You can create local views with the Create View statement. A view is like a named query. The view can also refer to any combination of other views.

Note: Local views are objects in the driver’s in-memory SQL Engine, not a native MongoDB collection. For information on native MongoDB views, see MongoDB views on page 100.

See "Supported SQL statements and extensions" for details on the Create View statement and other SQL statements supported by the driver.

See also

Supported SQL statements and extensions on page 161

MinKey and MaxKey values

The driver supports the MinKey and Maxkey special values and, by default, maps them to the `VARCHAR` ODBC data type during relational mapping. Note that MinKey and MaxKey values are static and return the same data whenever they are queried. The following are the respective values returned by the driver:

- MinKey: `{“$MinKey”:1}`
- MaxKey: `{“$MaxKey”:1}`

MinKey and MaxKey are special values used internally by MongoDB that do not have a counterpart in SQL. Therefore, they cannot be used in Where clauses for comparisons.

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.

```
[MongoDB]
Driver=ODBCHOME/lib/ivmongodb28.so
Description=My MongoDB Data Source
...
AuthenticationDatabase=mydb2
...
Database=MyDB
...
EnablePacketLogging=1
...
HostName=myserver
...
LogonID=JOHN
...
PacketLoggingFilePrefix=/tmp/myPacketLog
...
PacketLoggingMaxFileSize=102400
...
PacketLoggingMaxNumFiles=10
...
Password=secret
...
```

Connection string

You can specify connection options that configure packet logging in connection strings.

```
DRIVER=DataDirect 8.1 MongoDB;HostName=MyServer;AuthenticationDatabase=mydb2;
Database=MyDB;LogonID=JOHN;Password=secret;EnablePacketLogging=1;
PacketLoggingFilePrefix=C:\temp\myPacketLog;
```

Packet logging connection options

The following table describes the connection options used to configure packet logging.

Table 28: 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.

Option	Description
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 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.

6

Connection option descriptions

The connection option descriptions in this section 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 the option topics.

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 following tables 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 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 tables provide a summary of supported connection options by functionality, including their attribute names, short names, and default values.

- General options
- User ID and password authentication options
- Kerberos authentication options
- LDAP authentication options
- Data encryption options
- Mapping options
- SQL Engine options
- Proxy server options

- Additional Options

General options

The following table summarizes the general connection options that can apply to all connections that use data sources.

Table 29: General options

Attribute (Short Name)	Default
DatabaseName (db)	No default value
DataSourceName (dsn)	No default value
Description (n/a)	No default value
Hostname (host)	No default value
PortNumber (port)	`27017`

User ID and password authentication options

The following table summarizes the options used for user id and password authentication.

Table 30: User ID and password authentication options

Attribute (Short Name)	Default
AuthenticationMethod (am)	`0` (User ID and password)
AuthenticationDatabase (adb)	No default value
Hostname (host)	No default value
Password (pwd)	No default value
User	No default value

Kerberos authentication options

The following table summarizes the connection properties used for Kerberos authentication.

Table 31: Kerberos authentication options

Attribute (Short Name)	Default
AuthenticationMethod (am)	`0` (User ID and password)
ServicePrincipalName (spn)	No default value
User	No default value

LDAP authentication options

The following table summarizes the connection options used for LDAP authentication.

Table 32: LDAP authentication options

Attribute (Short Name)	Default
AuthenticationMethod (am)	0 (User ID and password)
Hostname (host)	No default value
Password (pwd)	No default value
User	No default value

Data encryption options

The following table summarizes the options used for TLS/SSL data encryption.

Table 33: Data encryption options

Attribute (Short Name)	Default
CryptoProtocolVersion (cpv)	No default value
EncryptionMethod (em)	0 (NoEncryption)
HostNameInCertificate (hnic)	No default value
KeyPassword (kp)	No default value
Keystore (ks)	No default value
KeystorePassword (ksp)	No default value
Truststore (ts)	No default value
TruststorePassword (tsp)	No default value
ValidateServerCertificate (vsc)	1 (true)

Mapping options

The following table summarizes the connection options involved in mapping the MongoDB data model to a relational model.

Table 34: Mapping options

Attribute (Short Name)	Default
ArrayNormalizationThreshold (art)	12 (elements)
ColumnDiscoverySampleSize (cdss)	1000

Attribute (Short Name)	Default
CreateMap (cm)	2 (NotExist)
FlattenArrayBase (fab)	1
JSONColumns (jcs)	0 (false)
KeywordConflictSuffix (kcs)	No default value
LeadingUnderscoreReplacement (lusr)	None. When no value is specified, a leading underscore is used in identifiers.
LegacyVirtualKeys (lvk)	0 (false)
QualifyNormalizedNames (qnn)	1 (Table)
RefreshSchema (rs)	0 (false)
SchemaFormat (sfmt)	2 (Mixed)
SchemaMap (smp)	Default value depends on environment
SpecialCharBehavior (scb)	0 (Strip)
UppercaseIdentifiers (uci)	1 (true)

SQL engine options

The following table lists the options used to configure the SQL engine.

Table 35: SQL engine options

Attribute (Short Name)	Default
BrokerIdleTimeout (bit)	600
BrokerPingInterval (bping)	30
BrokerPortNumber (bport)	1183
JVMArgs (jvma)	For the 32-bit driver when the SQL Engine Mode is set to 2 (Direct) or 3 (Broker): -Xmx256m For all other configurations: -Xmx1024m
JVMClassPath (jvmcp)	No default value

Attribute (Short Name)	Default
JVMPath (jvmp)	*install_dir*\jre\bin\java.exe
MaxConnectionsPerServer (mcps)	1024
ServerIdleTimeout (sito)	300
ServerLaunchTimeout (slt)	30
ServerPortNumber (spn)	32-bit: 19968 64-bit: 19967
SQLEngineMode (sem)	For Windows: 1 (Server) For UNIX/Linux: The SQL engine runs in Direct mode by default.
SQLService (ss)	No default value

Proxy server options

The following table summarizes the proxy server connection options.

Table 36: Proxy server options

Attribute (Short Name)	Default
ProxyHost (pxhn)	No default value
ProxyPassword (pxpw)	No default value
ProxyPort (pxpt)	0 which means that the default value is determined by whether the value specified for the Proxy Host (ProxyHost) option is an HTTP or HTTPS URL. For HTTP: 80 For HTTPS: 443
ProxyUser (pxun)	No default value

Additional options

The following table summarizes additional connection options supported by the driver.

Table 37: Additional options

Attribute (Short Name)	Default
ApplicationUsingThreads (aut)	1 (True)
EnableDNSLookup (edu)	1 (true)
ExtendedOptions (xo)	No default value
FetchSize (fs)	100 (rows)
InitializationString (is)	No default value
LogConfigFile (lgcf)	ddlogging.properties
NetworkMessageCompressors (nmc)	none
ReadOnly (ro)	0 (false)
ReadPreference (rp)	1 (Primary)
ReplicaSetName (rsn)	No default value
ReportCodepageConversionErrors (rcce)	0 (Ignore Errors)
ResultMemorySize (rms)	-1
SchemaFilter (sf)	No default value
TransactionMode (tm)	0 (NoTransactions)
VarcharThreshold (vth)	4000

For details, see the following topics:

- Application Using Threads
- Array Normalization Threshold
- Authentication Method
- Authentication Database
- Broker Idle Timeout
- Broker Ping Interval
- Broker Port Number
- Column Discovery Sample Size
- Create Map

- Crypto Protocol Version
- Data Source Name
- Database Name
- Description
- Enable DNS Lookup
- Encryption Method
- Extended Options
- Fetch Size
- Flatten Array Base
- Host Name
- Host Name in Certificate
- Initialization String
- JSON Columns
- JVM Arguments
- JVM Classpath
- JVM Path
- Key Password
- Keystore
- Keystore Password
- Keyword Conflict Suffix
- Leading Underscore Replacement
- Legacy Virtual Keys
- Log Config File
- Max Connections Per Server
- Min Varchar Size
- Network Message Compressors
- Password
- Port Number
- Proxy Host
- Proxy Password
- Proxy Port
- Proxy User
- Qualify Normalized Names

- Read Only
- Read Preference
- Refresh Schema
- Replica Set Name
- Report Codepage Conversion Errors
- Result Memory Size
- Schema Filter
- Schema Format
- Schema Map
- Server Idle Timeout
- Server Launch Timeout
- Server Port Number
- Service Principal Name
- Special Char Behavior
- SQL Engine Mode
- SQL Service
- String Truncation Method for Writes
- Timestamp Format
- Transaction Mode
- Truststore
- Truststore Password
- Uppercase Identifiers
- User
- Validate Server Certificate
- Varchar Threshold

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` (true), the driver works with single-threaded and multi-threaded applications.

If set to `0` (false), 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 Value

`1` (true)

See also

Performance considerations on page 85

Array Normalization Threshold

Attribute

ArrayNormalizationThreshold (art)

Purpose

Specifies the length of arrays (in elements) at which the driver begins to normalize elements in arrays to child tables when generating a flattened relational view (`SchemaFormat=Flatten`). In the flattened relational view, elements in arrays are typically flattened into columns in the parent table. This behavior can create tables with a high-number of columns when encountering large arrays or arrays nested in arrays. To avoid the memory and performance issues associated with handling very large tables, you can limit the size of the parent table using this option.

Valid Values

`0` | *`x`*

where:

`x`

is the length of arrays (in elements) at which the driver begins to normalize elements in arrays to child tables when generating a flattened view.

Behavior

If set to `0`, all elements in arrays are flattened as columns to the parent table.

If set to *`x`*, arrays containing a number of elements equal to or greater than the specified value are normalized to child tables when generating a flattened view. In arrays comprising of fewer elements than the specified amount, the elements are flattened into columns in the parent table.

Notes

- The behavior of this option also applies to nested arrays. Therefore, if the length of a nested array exceeds the value specified for this option, a separate child table will be generated for the elements of the nested array.

Default Value

`12`

Authentication Method

Attribute

AuthenticationMethod (am)

Purpose

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

Valid Values

`15` | `0` | `17` | `4`

Behavior

If set to `15` (None), the driver does not attempt to authenticate.

If set to `0` (UserIDPassword), the driver uses user ID/password authentication. The User option provides the user ID, and the Password option provides the password.

If set to `17` (Plain), the driver uses LDAP authentication. You must also provide a value for the User and Password options.

Important: When LDAP authentication is enabled, credentials are passed in clear text. Therefore, you should use LDAP authentication only on servers that are configured for TLS/SSL encryption.

If set to `4` (Kerberos), the driver uses Kerberos authentication.

Notes

- When `AuthenticationMethod=UserIdPassword`, it is recommended that you specify the database in which your user ID was created using the Authentication Database (AuthenticationDatabase) option. Since you can create the same user ID in different databases in MongoDB, this practice ensures that you are attempting to connect to the user ID with the correct permissions.
- When authentication is enabled, the client application will have access only to databases as dictated by the roles and privileges of the authenticated user.

Default Value

`0` (UserIDPassword)

See also

Authentication on page 75

Authentication Database

Attribute

AuthenticationDatabase (adb)

Purpose

Specifies the database in which your user ID was created. In MongoDB, you can create the same user ID in different databases with unique permissions. This option ensures that the driver authenticates with the correct version of the user ID when using user ID and password authentication (`AuthenticationMethod=userIdPassword`).

Valid Values

`string`

where:

`string`

is the name of the database in which your user ID was created.

Notes

- AuthenticationDatabase is used only for user ID and password authentication (`AuthenticationMethod=userIdPassword`). To ensure the correct permissions are used for your connection, it's recommended that you specify a value for AuthenticationDatabase when user ID and password authentication is enabled.
- If you do not specify a value for this option, the driver attempts to use your user ID with the database specified by the Databasename option. If your user ID was not created in the specified database, the driver will attempt to connect to the Admin database using your user ID and password.

Default Value

No default value

See also

User ID and password authentication on page 75

Broker Idle Timeout

Attribute

BrokerIdleTimeout (bit)

Purpose

Specifies the number of seconds the Broker continues to run after all the SQL engine servers have been stopped (shutdown). This option allows for the Broker to shut itself down while there are no active connections to the database. When a new connection request is made by the application, the Broker restarts itself and spawns a SQL engine process to serve the database requests made through the driver.

This option is used when the SQL Engine is operating in Broker mode.

Valid Values

timeout_seconds

where:

timeout_seconds

is the number of seconds the Broker continues to run after all the SQL engine servers have been stopped. This can be a value from `1` to `31622400`.

Notes

- This option is used when SQL Engine Mode (SQLEngineMode) is set to `3` (Broker).

Default Value

`600`

See also

Using the SQL engine server on page 87

Broker Ping Interval

Attribute

BrokerPingInterval (bping)

Purpose

Specifies the number of seconds between status heartbeats from a SQL engine server process to the Broker. Once the Broker stops receiving the heartbeat signal, it will begin the shutdown process.

This option is used when the SQL Engine is operating in Broker mode.

Valid Values

timeout_seconds

where:

timeout_seconds

is the number of seconds between status heartbeats from a SQL engine server process to the Broker. This can be a value from `1` to `3600`.

Notes

- This option is used when SQL Engine Mode (SQLEngineMode) is set to `3` (Broker).

Default Value

`30`

See also

Using the SQL engine server on page 87

Broker Port Number

Attribute

BrokerPortNumber (bport)

Purpose

Specifies a valid port on which the Broker listens for client connection requests. This option is used when the SQL Engine is operating in Broker mode.

Valid Values

port_number

where:

port_number

is the port number of the Broker listener.

Notes

- This option is used when SQL Engine Mode (SQLEngineMode) is set to `3` (Broker).

Default Value

`1183`

See also

Using the SQL engine server on page 87

Column Discovery Sample Size

Attribute

ColumnDiscoverySampleSize (cdss)

Purpose

Specifies the number of rows the driver fetches per collection when sampling data to detect columns and gather column statistics. The information collected in these samples is used when creating a schema map. Larger fetch sizes return samples that are more representative of your data, but at the expense of slower performance.

Valid Values

x

where:

x

specifies the number of rows the driver fetches per collection.

Notes

- The setting for this option is written to the configuration file when generating the schema map. If you attempt a subsequent connection using the configuration file and specify a different value for this option, the driver will return an error.

Default Value

`1000`

Create Map

Attribute

CreateMap (cm)

Purpose

Determines whether the driver creates the internal files required for a relational map of the native data when establishing a connection.

Valid Values

`0` | `1` | `2`

Behavior

If set to `0` (No), the driver uses the pre-existing group of schema map files specified by the Schema Map option. If the files do not exist, the connection fails.

If set to `1` (ForceNew), the driver replaces the schema map files specified by the Schema Map option with a newly generated group at the same location.

Note: The 8.1 driver uses an improved normalization schema to expose the JSON structure of MongoDB data as a relational view. The new schema map files use the 8.1 normalization format, which is not expected to be backwards compatible with SQL queries written for the 8.0 schema format. Queries will need to be revised to accommodate for the differences.

If set to `2` (NotExist), the driver uses the pre-existing group of schema map files specified by the Schema Map option. If the files do not exist, the driver creates them.

Notes

- When this option is set to `1` (ForceNew), the driver creates a backup of the 8.0 schema map files specified by the Schema Map option. At connection, if the driver detects a schema map file using the 8.0 format, it will rename the files using the following naming convention:

  ```
  <original_file_prefix>.<timestamp>.backup
  ```

The driver will then create a set of 8.1 schema map files using the location and or name specified by the Schema Map option. If no file name or prefix is specified, the driver will use the default file name.

- You can refresh the collections and fields exposed by the driver in an existing relational view of your data with the SQL extension Refresh Map. Refresh Map runs a discovery against your native data and adds newly discovered collections and newly discovered fields in existing collections to the schema map. Note that this process is additive only and will not delete any pre-existing collections or fields, even if they are no longer available for sampling in the MongoDB database.

Default Value

`2` (NotExist)

See also

Schema Map on page 148
Refresh Map (EXT) on page 166

Crypto Protocol Version

Attribute

CryptoProtocolVersion (cpv)

Purpose

Specifies a cryptographic protocol or comma-separated list of cryptographic protocols that can be used when SSL is enabled.

Valid Values

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

where:

cryptographic_protocol

is one of the following cryptographic protocols:

`TLSv1.3` | `TLSv1.2` | `TLSv1.1` | `TLSv1` | `SSLv3` | `SSLv2`

Note: The protocols available depend on your Java version. Most modern implementations have disabled all but TLSv1.3 and TLSv1.2.

Caution: To avoid vulnerabilities associated with older protocols, best security practices recommend using TLSv1.2 or higher.

Example

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

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

Notes

- When multiple protocols are specified, the driver uses the highest version supported by the server. If none of the specified protocols are supported by the server, the connection fails and the driver returns an error.
- When no value has been specified for this option, the driver establishes an SSL connection using the default. If the default is not supported by the server, the connection fails and the driver returns an error.
- The default may be set in the Java system property `https.protocols`, which is often set on the Java command line with the `-D`*`property`*`=` option. For example: `-Dhttps.protocols=TLSv1.3,TLSv1.2`

Default Value

No default value

See also

TLS/SSL encryption on page 79

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 Value

No default value

Database Name

Attribute

DatabaseName (db)

Purpose

Specifies the name of the database to which you are connecting. This value is used as the default qualifier for unqualified table names in SQL queries.

Valid Values

database_name

where:

database_name

is the name of a valid database.

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

Default Value

No default value

Description

Attribute

Description (desc)

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 Value

No default value

Enable DNS Lookup

Attribute

EnableDNSLookup (edu)

Purpose

Specifies whether the driver performs a DNS lookup to discover member nodes of a cluster when connecting. If the driver discovers cluster nodes in the specified domain, the driver will attempt to connect to an available node.

Valid Values

0 | 1

Behavior

If set to 1 (true), at connection, the driver performs a DNS lookup on the host specified by the Host Name (HostName) option. When a domain is specified by Host Name, the driver will attempt to discover the nodes of a cluster, then connect to an available node. If you are not connecting to a clustered environment, the driver will connect to the server specified by ServerName.

If set to 0 (false), the driver does not attempt to perform a DNS lookup at connection. Instead, it will connect directly to the host specified by the Host Name option. Specify this value if you are not connecting to clustered environment.

Notes

- Setting EnableDNSLookup to 1 (enabled) does not prohibit you from connecting to a non-clustered environment; however, the driver will still perform the lookup at connection. To improve connection time, you can disable the lookup (`EnableDNSLookup=0`) if you are not connecting to a clustered environment.

Default Value

1 (true)

See also

MongoDB Atlas clusters on page 82
Performance considerations on page 85

Encryption Method

Attribute

EncryptionMethod (em)

Purpose

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

Valid Values

0 | 1

Behavior

If set to 0 (NoEncryption), data is not encrypted or decrypted.

If set to 1 (SSL), data is encrypted using SSL. If the database server does not support SSL, the connection fails and the driver throws an exception.

Notes

When SSL is enabled, the following options also apply:

- CryptoProtocolVersion (CryptoProtocolVersion)

- Host Name In Certificate (HostNameInCertificate)
- Key Password (KeyPassword), for SSL client authentication
- Key Store (KeyStore), for SSL client authentication
- Key Store Password (KeyStorePassword), for SSL client authentication
- Trust Store (TrustStore)
- Trust Store Password (TrustStorePassword)
- Validate Server Certificate (ValidateServerCertificate)

Default Value

`0` (NoEncryption)

See also

Performance considerations on page 85
TLS/SSL encryption on page 79

Extended Options

Attribute

ExtendedOptions (xo)

Purpose

Specifies a semicolon separated list of connection options and their values. Use this connection option to set the value of undocumented connection options that are provided by Progress DataDirect Technical Support.

Valid Values

option=value[;*option=value*;...]

where:

option

is a connection option.

value

is the value or setting of the connection option.

Default Value

No default value

Fetch Size

Attribute

FetchSize (fs)

Purpose

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

Valid Values

`0` | *x*

where:

x

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

Behavior

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

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

Notes

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

Default Value

`100`

See also

Performance considerations on page 85

Flatten Array Base

Attribute

FlattenArrayBase (fab)

Purpose

Specifies the starting ordinal value appended to column names for flattened arrays. When flattening arrays, column names are appended with an underscore and the ordinal value (`<array_name>_<ordinal_location>`).

Valid Values

`0` | `1`

Behavior

If set to `0`, the first column name in the array will be appended with an _0. For example, the first three columns generated in the vehicles array would be the following in the Mixed relational view: `VEHICLES_0`, `VEHICLES_1`, `VEHICLES_2`.

If set to `1`, the first column name in the array will be appended with an _1. For example, the first three columns generated in the vehicles array would be the following in the Mixed relational view: `VEHICLES_1`, `VEHICLES_2`, `VEHICLES_3`.

Default Value

`1`

Host Name

Attribute

HostName (host)

Purpose

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

Valid Values

string

where:

string

is a valid IP address or server name.

Default Value

None

Host Name in Certificate

Attribute

HostNameInCertificate (hnic)

Purpose

Specifies a host name for certificate validation when SSL encryption is enabled and validation is enabled (`ValidateServerCertificate=1`). This option is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

Valid Values

`host_name`

where:

`host_name`

is a valid host name.

Behavior

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

Notes

- If SSL encryption or certificate validation is not enabled, this option is ignored.
- If SSL encryption and validation is enabled and this option is unspecified, the driver uses the value of the Server Name (ServerName) option to validate the certificate.

Default Value

No default value

See also

TLS/SSL encryption on page 79

Initialization String

Attribute

InitializationString (is)

Purpose

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

Valid Values

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

where:

command

is a SQL command.

Notes

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

Default Value

No default value

JSON Columns

Attribute

JSONColumns (jcs)

Purpose

Determines whether the driver exposes documents and arrays embedded within a collection as JSON formatted fields, in addition to exposing individual collection and array elements as fields, when mapping to a flattened view (`SchemaFormat=1`). Exposing documents and arrays as JSON values can make certain operations more convenient, such as when tools use this data as a single object for communication. In addition, in rare instances where variations in data structure might cause this data to not be sampled, enabling this option provides a method in which the data can still be read as a JSON value.

Valid Values

`0` | `1`

Behavior

If set to `1` (true), the driver exposes documents and arrays embedded within a collection as JSON formatted fields. In addition, the individual collections and arrays are also exposed as fields in accordance to the flattened view.

If set to `0` (false), the driver exposes documents and arrays embedded within a collection according to the relational view specified by the Schema Format (SchemaFormat) option.

Notes

- Querying JSON values can be an expensive operation that could negatively impact performance; therefore, you should only query JSON values when necessary.

Default Value

`0` (false)

JVM Arguments

Attribute

JVMArgs (jvma)

Purpose

A string that contains the arguments that are passed to the JVM that the driver is starting. The location of the JVM is specified using the JVM Path (JVMPath) option.

When specifying the heap size for the JVM, the JVM tries to allocate the heap memory as a single contiguous range of addresses in the application's memory address space. If the application's address space is fragmented so that there is no contiguous range of addresses big enough for the amount of memory specified for the JVM, the driver fails to load, because the JVM cannot allocate its heap. This situation is typically encountered only with 32-bit applications, which have a much smaller application address space. If you encounter problems with loading the driver in an application, try reducing the amount of memory requested for the JVM heap. If possible, switch to a 64-bit version of the application.

Valid Values

String

where:

String

contains arguments that are defined by the JVM. Values that include special characters or spaces must be enclosed in curly braces `{ }` when used in a connection string.

Example

To set the heap size used by the JVM to 256 MB and the http proxy information, specify:

```
{-Xmx256m -Dhttp.proxyHost=johndoe -Dhttp.proxyPort=808}
```

To set the heap size to 256 MB and configure the JVM for remote debugging, specify:

```
{-Xmx256m -Xrunjdwp:transport=dt_socket, address=9003,server=y,suspend=n -Xdebug}
```

Default Value

For the 32-bit driver when the SQL Engine Mode connection option is set to `2` (Direct) or `3` (Broker):

`-Xmx256m`

For all other configurations:

`-Xmx1024m`

See also

Performance considerations on page 85

JVM Classpath

Attribute

JVMClassPath (jvmcp)

Purpose

Specifies the CLASSPATH for the Java Virtual Machine (JVM) used by the driver. The CLASSPATH is the search string the JVM uses to locate the Java jar files the driver needs.

Valid Values

string

where:

string

specifies the CLASSPATH. Separate multiple jar files by a semi-colon on Windows platforms and by a colon on UNIX/Linux platforms. CLASSPATH values with multiple jar files must be enclosed in curly braces { } when used in a connection string.

If your process employs multiple drivers that use a JVM, the value of the JVM Classpath for all affected drivers must include an absolute path to all the jar files for drivers used in that process. In addition, the value specified must be identical for all drivers. For example if you are using the MongoDB driver and Autonomous REST Connector on Windows, you would specify a value of `{c:\install_dir\java\lib\mongodb.jar; c:\install_dir\java\lib\autorest.jar}` for both drivers. If the value for any of the affected drivers is missing a file path or is different from the one specified for the other drivers, the drivers will return an error at connection that the JVM is already running.

Example

On Windows:

```
{.;c:\install_dir\java\lib\mongodb.jar}
```

On UNIX/Linux:

```
{.:/home/user1/install_dir/java/lib/mongodb.jar}
```

Default Value

The default is an empty string, which means that the driver automatically detects the CLASSPATHs for all ODBC drivers installed on your machine and specifies them when launching the JVM.

JVM Path

Attribute

JVMPath (jvmp)

Purpose

Specifies fully qualified path to the JVM executable that you want to use to run the SQL Engine Server. The path must not contain double quotation marks.

Valid Values

`String`

where:

`String`

The full path to the JVM executable.

Default Value

`install_dir\jre\bin\java.exe`

Key Password

Attribute

KeyPassword (kp)

Purpose

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

Valid Values

`string`

where:

`string`

is a valid password.

Default Value

No default value

See also

TLS/SSL encryption on page 79

Keystore

Attribute

Keystore (ks)

Purpose

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

Valid Values

keystore_directory

where:

keystore_directory

is a valid directory of a keystore file.

Notes

- This value overrides the directory of the keystore file that is specified by the `javax.net.ssl.keyStore` Java system property. If this property is not specified, the keystore directory is specified by the `javax.net.ssl.keyStore` Java system property.
- The keystore and truststore files can be the same file.

Default Value

No default value

See also

TLS/SSL encryption on page 79

Keystore Password

Attribute

KeystorePassword (ksp)

Purpose

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

Valid Values

string

where:

string

is a valid password.

Notes

- This value overrides the password of the keystore file that is specified by the `javax.net.ssl.keyStorePassword` Java system property. If this property is not specified, the keystore password is specified by the `javax.net.ssl.keyStorePassword` Java system property.
- The keystore and truststore files can be the same file; therefore, they may have the same password.

Default Value

No default value

See also

TLS/SSL encryption on page 79

Keyword Conflict Suffix

Attribute

KeywordConflictSuffix (kcs)

Purpose

Specifies a string of up to 5 alphanumeric characters that the driver appends to any object or field name that conflicts with a SQL engine keyword.

Valid Values

`String`

where:

`String`

is a string of up to 5 alphanumeric characters.

Example

A field called `CASE` exists in the data schema. To avoid a naming conflict with the SQL engine keyword CASE, you could set `KeywordConflictSuffix=TAB`. In this scenario, the driver maps the `CASE` field to the `CASETAB` column.

Notes

- The setting for this option is written to the configuration file when generating the schema map. If you attempt a subsequent connection using the configuration file and specify a different value for this option, the driver will return an error.

Default Value

No default value

See also

Using identifiers on page 96

Leading Underscore Replacement

Attribute

LeadingUnderscoreReplacement (lusr)

Purpose

Specifies the string of characters that replace leading underscores used in identifiers for documents and arrays.

Valid Values

`string`

where:

`string`

is comprised of any Unicode character or group of characters, including spaces.

Example

MongoDB collections automatically include the _id field. By specifying `LeadingUnderscoreReplacement=XX`, the `_id` field becomes the `XXID` column in the relational view of the data. In addition, any other fields with a leading underscore would be modified in the same manner.

Notes

- The setting for this option is written to the configuration file when generating the schema map. If you attempt a subsequent connection using the configuration file and specify a different value for this option, the driver will return an error.

Default Value

None. When no value is specified, a leading underscore is used in identifiers.

See also

Using identifiers on page 96

Legacy Virtual Keys

Attribute

LegacyVirtualKeys (lvk)

Purpose

Specifies whether the driver generates legacy virtual keys for newly-discovered nested objects when mapping the relational view of data.

For versions earlier than 8.1, the driver used the naming convention *`object_name`*`_GENERATED_ID` for the unique virtual key column, which was used as a foreign key to associate the child table back to the parent table. Starting in version 8.1, the driver uses the `POSITION` column for this purpose.

By default (`LegacyVirtualKeys=0`), normalized schemas migrated from the 8.0 format retain the legacy key column for existing objects, while the driver generates `POSITION` columns for newly-discovered nested objects. To use a consistent naming convention for virtual key columns in migrated schemas, set this option to `1`.

Valid Values

`0` | `1`

Behavior

If set to `1` (true), the driver generates legacy virtual keys and `POSITION` columns when mapping newly-discovered nested objects for both new and migrated schemas. Virtual keys are populated in the *object_name*`_GENERATED_ID` column.

If set to `0` (false), the driver uses the `POSITION` column when mapping newly-discovered objects for both new and migrated schemas. Note that migrated schemas will continue to use the *object_name*`_GENERATED_ID` column for existing objects, but newly-discovered objects will have only the `POSTION` column.

Default Value

`0` (false)

Log Config File

Attribute

LogConfigFile (lgcf)

Purpose

Specifies the file name, and optionally, the path of the properties file used to initialize driver logging.

Valid Values

String

where:

String

is the relative or fully qualified path of the properties file to load to initialize driver logging. If you do not specify a path, the driver looks for this file in the current working directory. If the specified file does not exist, the driver continues searching for an appropriate properties file as described in "Logging for Java components" in the *Progress DataDirect for ODBC Drivers Reference*.

Default Value

`ddlogging.properties`

Max Connections Per Server

Attribute

MaxConnectionsPerServer (mcps)

Purpose

Specifies the number of connections that can be granted to a SQL engine server before the Broker creates another server. This option can be tuned to balance performance versus memory consumption. You can improve performance by limiting the number of connections handled by a single server; however, this is done at the expense of allocating more memory for additional servers.

This option is used when the SQL Engine is operating in Broker mode.

Valid Values

max_connections

where:

max_connections

is the maximum number of connections that can be granted to a server before another server is created. This can be a value from 2 to 1024.

Notes

- This option is used when SQL Engine Mode (SQLEngineMode) is set to 3 (Broker).

Default Value

1024

See also

Using the SQL engine server on page 87

Min Varchar Size

Attribute

MinVarcharSize (mnvs)

Purpose

Specifies the minimum default length, in characters, of fields that are mapped as VARCHAR. The default length of VARCHAR columns is 1.5 times the largest data value the driver samples from the column. When the default length is less than the value specified for this option, the driver increases the default length to the value set for this option. Setting this option to a larger value than the calculated default allows for larger data values to be inserted by the driver.

Valid Values

x

where:

x

is the maximum size of the default length in characters given to columns that are mapped as VARCHAR.

Example

For example, Min Varchar Size is set to `1000`, but the driver calculates a default length to be 500 based on the values sampled in your MongoDB STRING data column. Since the calculated value would be less than the setting of Min Varchar Size, the length is set to 1000. Conversely, in another column, the driver samples larger values and calculates a default length of 3000. Because the calculated value exceeds that of the Min Varchar Size option, the length would be set to 3000.

Notes

- The String Truncation Method For Writes (StringTruncationMethodForWrites) option determines the behavior of the driver when inserting String values that exceed the column length defined in the relational schema.

Default Value

`1`

Network Message Compressors

Attribute

NetworkMessageCompressors (nmc)

Purpose

Specifies whether the driver attempts to use data compression for all messages passed between the client and server. Data compression can significantly reduce network traffic, which, in turn, can lower data transfer costs for cloud services.

Valid Values

`none` | `zlib`

Behavior

If set to `none`, the driver passes messages between the client and server without attempting to use compression.

If set to `zlib`, messages passed between the client and the server are compressed using the zlib compression algorithm. If the server is not configured to use this type of compression for network messages, the driver falls back to passing messages without using compression.

Default Value

`none`

Password

Attribute

Password (pwd)

Purpose

A password that is used to connect to the server.

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

Behavior

password

where:

password

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

Default Value

No default value

See also

Authentication on page 75

Port Number

Attribute

PortNumber (port)

Purpose

Specifies the port number of the server listener.

Valid Values

port_number

where:

port_number

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

Default Value

`27017`

Proxy Host

Attribute

ProxyHost (pxhn)

Purpose

Identifies a proxy server to use for the first connection.

Valid Values

server_name | *IP_address*

where:

server_name

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

IP_address

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

Default Value

No default value

See also

Proxy server on page 82

Proxy Password

Attribute

ProxyPassword (pxpw)

Purpose

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

Valid Values

password

where:

password

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

Default Value

No default value

See also

Proxy server on page 82

Proxy Port

Attribute

ProxyPort (pxpt)

Purpose

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

Valid Values

port

where:

port

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

Default Value

`0` which means that the default value is determined by whether the value specified for the Proxy Host (ProxyHost) option is an HTTP or HTTPS URL.

For HTTP: `80`

For HTTPS: `443`

See also

Proxy server on page 82

Proxy User

Attribute

ProxyUser (pxun)

Purpose

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

Valid Values

user_name

where:

user_name

is a valid user ID for the proxy server.

Default Value

No default value

See also

Proxy server on page 82

Qualify Normalized Names

Attribute

QualifyNormalizedNames (qnn)

Purpose

Determines whether the names of relational child-tables normalized from arrays, objects, and subdocuments are prefixed with the collection name and any parent objects.

Valid Values

`0` | `1` | `2`

Behavior

If set to `0` (No), the relational table name is derived solely from the column name of the array, object, or subdocument.

If set to `1` (Table), the relational table name is prepended with the name of the collection. For example, if the name of the collection was named `Books` and the array column was named `Chapters`, then the relational table name would be `BOOKS_CHAPTERS`.

If set to `2` (FullPath), the relational table name is prepended with the names of all objects in which the array, object, or subdocument is nested. For example, if the collection was `BOOKS` with an array `Chapters` that contained an array `Pages`, then the resulting relational table name would be `BOOKS_CHAPTERS_PAGES`.

Notes

- If a naming conflict occurs, the driver appends an underscore separator and integer (for example, `_1`) to the table name.
- The value of this option also controls the name of the foreign key column in the child table. For example, if this option is set to `Table`, the foreign key column name would be `_ID` prepended with the parent object name. Therefore, a parent object of `BOOKS` would result in a foreign key column named `BOOKS_ID` in the child table.

Default Value

`1` (Table)

See also

Using identifiers on page 96

Read Only

Attribute

ReadOnly (ro)

Purpose

Specifies whether the connection has read-only access to the data source.

Valid Values

`0` | `1`

Behavior

If set to `1` (true), the connection has read-only access. The following commands are the only commands that you can use when a connection is read-only:

- Explain Plan
- Select (except Select Into)
- Set Schema

The driver returns an error if any other command is used.

If set to `0` (false), the connection is opened for read/write access, and you can use all commands supported by the product.

Caution:Before disabling the ReadOnly connection property, it is critical to confirm that all values in the primary key column are unique. Executing write operations against data with duplicate primary keys can produce unpredictable and undesirable results.

Default Value

`1` (true)

Read Preference

Attribute

ReadPreference (rp)

Purpose

Specifies a preference for the type of member (server node) of a replica set to which the driver attempts to connect. Connections to the primary member (read-write server nodes) return the most recent version of the data when executing Select queries, but increase the workload of the primary member and may negatively affect performance. To reduce the demand on the primary member, secondary members (read only server nodes) can be used at the expense of reading stale data.

Valid Values

0 | 1 | 2 | 3 | 4

Behavior

If set to 0 (None), the driver attempts to connect to only the member authorized by the application.

If set to 1 (Primary), the driver attempts to connect to only the primary member of a replica set. If the primary member is unavailable, the connection fails.

If set to 2 (PrimaryPreferred), the driver attempts to connect to the primary member first; but if it is unavailable, the driver attempts to connect to secondary members.

If set to 3 (Secondary), the driver attempts to connect to only secondary members of a replica set. If the secondary members of the replica set are unavailable, the connection fails.

If set to 4 (SecondaryPreferred), the driver attempts to connect to secondary members first; but if they are unavailable, the driver attempts to connect to the primary member.

Notes

- When connected to secondary members (read-only) of a replica set, the driver will return an error when attempting to execute write operations. You can work around this limitation by providing a value for the Replica Set Name (ReplicaSetName) option to enable failover for write operations.
- If the Read Preference option is configured to connect to a secondary member, the replica set that the driver is connecting to must contain a secondary member. If no secondary member exists, the driver will return an error.
- This option only affects the member to which the driver connects and only during the connection process. For example, if you specified a value of secondary, the driver will not attempt to identify a new secondary member if the secondary member it is connected to is promoted to the primary member.

Default Value

1 (Primary)

See also

Replica Set Name on page 143

Refresh Schema

Attribute

RefreshSchema (rs)

Purpose

Specifies whether the driver adds newly discovered objects to the relational map when connecting.

Valid Values

0 | 1

Behavior

If set to `1` (true), the driver adds newly discovered objects to the relational view of your data. At connection, the driver compares the relational map to a new sample of the data from the server. Any new objects that are detected are mapped to the relational view.

If set to `0` (false), the driver does not refresh the relational map when connecting to the server.

Notes

- This option is equivalent to executing the Refresh Map statement.

Default Value

`0` (false)

Replica Set Name

Attribute

ReplicaSetName (rsn)

Purpose

Specifies the name of the replica set against which the driver executes write operations. Providing a value for this option enables replica set failover for write operations.

Valid Values

replica_set_name

where:

replica_set_name

is the name of the replica set against which the driver performs write operations.

Notes

- When a value is specified for this option, the driver attempts to establish a connection to the primary node when a write operation is executed. If the primary node is unavailable, the driver repeats the discovery process until it finds the newly elected one or until the maximum number of retry attempts is met.

Default Value

No default value

See also

Replica set failover for write operations on page 84

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 server 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` (IgnoreErrors), the driver substitutes `0x1A` for each character that cannot be converted and does not return a warning or error.

If set to `1` (ReturnError), the driver returns an error instead of substituting 0x1A for unconverted characters.

If set to `2` (ReturnWarning), the driver substitutes `0x1A` for each character that cannot be converted and returns a warning.

Default Value

`0`

Result Memory Size

Attribute

ResultMemorySize (rms)

Purpose

Specifies the maximum size, in megabytes, of an intermediate result set that the driver holds in memory. When this threshold is reached, the driver writes a portion of the result set to disk in temporary files.

Valid Values

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

where:

x

is the maximum size, in MB, of a intermediate result set that the driver holds in memory.

Behavior

If set to `-1`, the maximum size of an intermediate result that the driver holds in memory is a percentage of the max Java heap size. When this threshold is reached, the driver writes a portion of the result set to disk.

If set to `0`, the driver holds the entire intermediate result set in memory regardless of size. No portion of the result set is written to disk. Setting ResultMemorySize to `0` can improve performance for result sets that easily fit within the JVM's free heap space, but can diminish performance for result sets that barely fit within the JVM's free heap space.

If set to *x*, the driver holds intermediate results in memory that are no longer than the size specified. When this threshold is reached, the driver writes a portion of the result set to disk.

Notes

- By default, Result Memory Size is set to `-1`. When set to `-1`, the maximum size of an intermediate result that the driver holds in memory is a percentage of the max Java heap size. When processing large sets of data, out-of-memory errors can occur when the size of the result set exceeds the available memory allocated to the JVM. In this scenario, you can tune ResultMemorySize to suit your environment. To begin, set ResultMemorySize equal to the max Java heap size divided by 4. Proceed by decreasing the value until out-of-memory errors are eliminated. As a result, the maximum size of an intermediate result set the driver holds in memory will be reduced, and some portion of the result set will be written to disk. Be away that while writing to disk reduces the risk of out-of-memory errors, it also negatively impacts performance. For optimal performance, decrease this value only to a size necessary to avoid errors.
- You can adjust the max Java heap size to address memory and performance concerns. By increasing the max Java heap size, you increase the amount of data the driver accumulates in memory. This can reduce the likelihood of out-of-memory errors and improve performance by ensuring that result sets fit easily within the JVM's free heap space. In addition, when a limit is imposed by setting ResultMemorySize to `-1`, increasing the max Java heap size can improve performance by reducing the need to write to disk, or removing it altogether.
- The Fetch Size (FetchSize) connection option can also be used to reduce demands on memory and decreasing the likelihood of out-of-memory errors.

Default Value

`-1`

See also

Fetch Size on page 124
Performance considerations on page 85

Schema Filter

Attribute

SchemaFilter (sf)

Purpose

Specifies a comma-separated list of database and collection pairs for which you want the driver to fetch metadata. This option can significantly improve connection times by limiting the collections for which metadata is fetched to only those that are required by your application. If you do not specify a value, the driver fetches metadata for all the collections in every database that your account has access to, which can adversely impact performance during the initial connection to a database.

Valid Values

database_name`:`*collection_name*[[`,*database_name*:`*`collection_name`*]...]

where:

database_name

is a literal value or regular expression for the database that contains collections for which the driver fetches metadata.

collection_name

is a literal value or regular expression of the collection for which the driver fetches metadata.

A schema or table name value can be:

- A literal name that does not contain either a comma (,) or a colon (:)
- A literal name that contains a comma (,) or a colon (:) that is bounded by a slash (/) at the beginning and end. For example, a collection named `sales:2019` would be represented by `/sales:2019/`
- A regular expression bounded by a slash (/) at the beginning and end, such as `/sales.*\d/`
- An asterisk (*), which represents all databases or collections within the corresponding schema(s).

Example

Literal values: The following example returns metadata for only the `november` and `march` collections in the `oem_sales` database.

```
SchemaFilter=oem_sales:november,oem_sales:march
```

Wildcard values: If you want the driver to fetch metadata for all the tables in a schema, replace the value for the table or schema name with an * (wildcard) character. For example, the following returns metadata for all the tables in the oem_sales and for tables named customers in all databases.

```
SchemaFilter=oem_sales:*,*:customers
```

Partial wildcard values: You can also use the asterisk to specify partial values for databases and collections. For example, the following returns metadata for all tables that end with region that are in schemas that begin with sales.

```
SchemaFilter=/sales.*/:/.*region/
```

Regular expressions: The following returns metadata for tables named tax in all databases that start with year which end with a number.

```
SchemaFilter=*/year.*\d/:tax
```

Notes

- The setting for this option is written to the configuration file when generating the schema map. If you attempt a subsequent connection using the configuration file and specify a different value for this option, the driver will return an error.

Default Value

No default value

See also

Performance considerations on page 85

Schema Format

Attribute

SchemaFormat (sfmt)

Purpose

Specifies to which model of the relational view the driver maps data.

Valid Values

`0` | `1` | `2`

Behavior

If set to `0` (NormalizeAll), the driver generates a normalized relational view of your data, mapping subdocuments and arrays as child tables with foreign key relationships to parent tables.

If set to `1` (Flatten), the driver generates a flattened relational view of your data, mapping subdocuments and arrays as columns within a single, comprehensive relational table for each collection.

If set to `2` (Mixed), the driver generates a normalized view of data that flattens certain objects into columns in the parent table. For example, subdocuments are mapped as columns in the parent table, while arrays and nested subdocuments are mapped to separate child tables. Refer to the "Mapping objects to tables" for details on how the mixed schema format normalizes or flattens subdocuments and arrays.

Notes

- The setting for this option is written to the configuration file when generating the schema map. If you attempt a subsequent connection using the configuration file and specify a different value for this option, the driver will return an error.

Default Value

`2` (Mixed)

See also

Mapping objects to tables on page 34

Schema Map

Attribute

SchemaMap (smp)

Purpose

Specifies the fully qualified path of the configuration file where the relational map of native data is written. The driver looks for this file when connecting to a MongoDB server. The driver generates the files based on the setting of the Create Map (CreateMap) option.

Valid Values

string

where:

string

is the absolute path and filename of the configuration file, including the `.config` extension. For example, if specifying a value of:

`C:\\Users\\Default\\AppData\\Local\\Progress\\DataDirect\\MongoDB_Schema\\MyServer.config`

the driver either creates or looks for the configuration file Myserver.config in the following directory:

`C:\Users\Default\AppData\Local\Progress\DataDirect\MongoDB_Schema\`

Notes

- When connecting to a server, the driver looks for the schema map configuration file. If the configuration file does not exist, the driver creates a schema map using the name and location you have provided. If you do not provide a name and location for the schema map, the driver creates a schema map using default values.
- The driver uses the path specified in this connection option to store additional internal files.
- The SchemaDefinition attribute is an alias for the Schema Map (SchemaMap) option.

Default Value

The default is determined by the environment. The driver attempts to create the files in a subdirectory of the first available directory in the following order:

- Windows
  - DD_HOME environment variable
  - LOCALAPPDATA environment variable
  - APPDATA environment variable
  - User’s home directory

For Windows, the the file path takes the following format:

```
<available_location>\Progress\DataDirect\MongoDB_Schema\<user_name>.config
```

- For UNIX/Linux
  - DD_HOME environment variable

- User's home directory

For UNIX/Linux, the file path takes the following format:

```
<available_location>/progress/datadirect/MongoDB_schema/<user_name>.config
```

Server Idle Timeout

Attribute

ServerIdleTimeout (sito)

Purpose

Specifies the number of seconds the SQL engine server can be without active connections before the Broker issues a `SHUTDOWN` command to stop the server. This option can be used to manage the memory and resource consumption by shutting down the server when it is not in use.

This option is used when the SQL Engine is operating in Broker mode.

Valid Values

timeout_seconds

where:

timeout_seconds

is the number of seconds the SQL engine server can be without active connections before the Broker stops the server process. This can be a value from `1` to `3162240`.

Notes

- This option is used when SQL Engine Mode (SQLEngineMode) is set to `3` (Broker).

Default Value

`300`

See also

Using the SQL engine server on page 87

Server Launch Timeout

Attribute

ServerLaunchTimeout (slt)

Purpose

Specifies the number of seconds the Broker waits for a server process to initialize before determining that is not available. The Broker will attempt to launch a server process five times before failing the connection attempt.

Valid Values

timeout_seconds

where:

timeout_seconds

is the number of seconds the Broker waits for a server process to initialize before determining it is not available. This can be a value from 1 to 3600.

Notes

- This option is used when SQL Engine Mode (SQLEngineMode) is set to 3 (Broker).

Default Value

30

See also

Using the SQL engine server on page 87

Server Port Number

Attribute

ServerPortNumber (sport)

Purpose

Specifies a valid port on which the SQL engine listens for requests from the driver.

Valid Values

port_number

where:

port_number

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

Notes

- This option is ignored when SQL Engine Mode (SQLEngineMode) is set to 2 (Direct).

Default Value

32-bit: 19968

64-bit: 19967

See also

Using the SQL engine server on page 87

Service Principal Name

Attribute

ServicePrincipalName (spn)

Purpose

Specifies the three-part service principal name registered with the key distribution center (KDC) in a Kerberos configuration.

Valid Values

Service_Name/Fully_Qualified_Domain_Name@REALM_NAME

where:

Service_Name

is the name of the service hosting the instance. The default value is `mongodb`.

Fully_Qualified_Domain_Name

is the fully qualified domain name (FQDN) of the host machine. By default, the driver uses the value specified by the Host Name (HostName) option. This value must match the FQDN registered with the KDC. The FQDN consists of a host name and a domain name. For the example `myserver.test.com`, `myserver` is the host name and `test.com` is the domain name.

REALM_NAME

is the name of the Kerberos realm. By default, the driver uses the default realm specified in the Kerberos configuration file. This part of the value must be specified in upper-case characters, for example, `EXAMPLE.COM`. For Windows Active Directory, the Kerberos realm name is the Windows domain name.

Example

The following is an example of a valid service principal name.

```
mongodb/myserver.test.com@EXAMPLE.COM
```

Notes

- By default, the driver builds the Service Principal Name by concatenating the service name `mongodb`, the FQDN as specified with the Host Name option, and the default realm name as specified in the Kerberos configuration file. If this value does not match the service principal name registered with the KDC, then the value of the service principal name registered with the KDC should be specified for the Service Principal Name option.
- In a Kerberos configuration, an IP address cannot be used as a FQDN.
- If Authentication Method (AuthenticationMethod) is set to `UserIdPassword`, the value of the Service Principal Name option is ignored.

Default Value

No default value

Special Char Behavior

Attribute

SpecialCharBehavior (scb)

Purpose

Determines how the driver handles the mapping of native identifiers containing characters that would require them to be quoted in SQL statements.

Valid Values

`0` | `1` | `2`

Behavior

If set to `0` (Strip), the driver removes any characters that are not part of a legal, unquoted SQL identifier. In practice, this removes any characters that are not letters, digits, or underscores. For example, if the native name were `Cost of Customer Acquisition`, the mapped name would be `CostofCustomerAcquisition`.

If set to `2` (Replace), the driver replaces any characters that would cause the identifier to be quoted with underscores. For example, if the native name was `Yearly Cost Percentage`, the mapped name would be `Yearly_Cost_Percentage`.

If set to `1` (Include), the driver does not modify native identifiers; therefore, identifiers containing characters that are not letters, digits, or underscores would need to be quoted. For example:

```
SELECT "Long & (Very) Unusual Field/Name" FROM "Oddly-Named+Table"
```

Default Value

`0` (Strip)

See also

Using identifiers on page 96

SQL Engine Mode

Attribute

SQLEngineMode (sem)

Purpose

Specifies whether the driver's SQL engine runs in the same process as the driver (direct mode), runs in a process that is separate from the driver (server mode), or runs in a separate process that is monitored by the driver's Broker (Broker mode). You must be an administrator to modify the server mode configuration values, and to start or stop the SQL engine service.

Valid Values

0 | 1 | 2 | 3

Behavior

If set to 0 (Auto), the SQL engine attempts to run in server mode first; however, if server mode is unavailable, it runs in direct mode. To use server mode with this value, you must start the SQL Engine service before using the driver (see "Using the SQL engine server" for more information).

If set to 1 (Server), the SQL engine runs in server mode. The SQL engine operates in a separate process from the driver within its own JVM. You must start the SQL Engine service before using the driver (see "Using the SQL engine server" for more information).

If set to 2 (Direct), the SQL engine runs in direct mode. The driver and its SQL engine run in a single process within the same JVM.

Important: Changes you make to the server mode configuration affect all DSNs sharing the service.

If set 3 (Broker), the SQL engine runs in Broker mode. The SQL engine operates in an external Java process that is monitored by the driver's Broker. To efficiently provisions memory and resources, the Broker automatically starts and stops the service as needed. This setting also eliminates the need to manually start and stop the service.

Default Value

For Windows: 1 (Server)

For UNIX/Linux: 2 (Direct)

See also

Using the SQL engine server on page 87

SQL Service

Attribute

SQLService (ss)

Purpose

Displays the name of the ODBC SQL engine service that runs as a separate process instead of being loaded within the process of an ODBC application.

Note: This option is used only for display purposes in the configuration manager. No value should be specified for this option.

Default Value

No default value

See also

Using the SQL engine server on page 87

String Truncation Method for Writes

Attribute

StringTruncationMethodForWrites (stmfw)

Purpose

Determines the behavior of the driver when attempting to insert String values that exceed the column length defined in the relational schema.

Valid Values

`0` | `1`

Behavior

If set to `0` (Keep), the driver inserts String values that exceed the column length defined in the relational schema without truncating the value.

If set to `1` (Error) the driver returns an error when attempting to insert a String value that exceeds the column length defined in the relational schema.

Notes

- For STRING columns, the driver determines the column size to be 50% greater than the longest sampled value in the column. Setting this option to `0` (keep) allows for inserts with values larger than those identified through sampling to succeed.

Default Value

`1` (Error)

Timestamp Format

Attribute

TimestampFormat (tsf)

Purpose

Specifies the format in which the driver renders the MongoDB composite timestamp. This option allows you to specify the format of the MongoDB composite timestamp that is most appropriate for your use case.

Valid Values

`0` | `1` | `2` | `3` | `4`

Behavior

If set to `0` (hex), the driver renders the bytes comprising the timestamp value as a string of hexadecimal digits, echoing the storage format. For example, `000000001407AC1A`.

If set to `1` (bigint), the driver renders the timestamp as a 64-bit integer. This setting is optimal if timestamps are used for ordering the result set in chronological order. For example, `1921918923461099520`.

If set to, `2` (bson), the driver renders the value in the format used for serializing as BSON, as is used in the MongoDB shell. The format is the expression:

```
Timestamp(<t>, <i>)
```

where `<t>` is the number of seconds since the Unix epoch, and `<i>` is the sequence number. For example, `Timestamp(447481620, 0)`.

If set to `3` (json), the driver renders the value in the format used for serializing as JSON. The format is:

```
{"t":<t>, "i":<i>}
```

where `<t>` is the number of seconds since the Unix epoch, and `<y>` is the sequence number. For example, `{"t":447481620, "i":0}`.

If set to `4` (text), the driver renders the timestamp as a pair of values. The first is the date and time portion. The second is the sequence number. For example, `1984-03-07T04:27:00, 0`.

Default Value

`1` (bigint)

Transaction Mode

Attribute

TransactionMode (tm)

Purpose

Specifies how the driver handles manual transactions.

Valid Values

`0` **|** `1` **|** `3`

Behavior

If set to `0` (NoTransactions), the data source and the driver do not support transactions. Metadata indicates that the driver does not support transactions.

If set to `1` (Ignore), the data source does not support transactions and the driver always operates in auto-commit mode. Calls to set the driver to manual commit mode and to commit transactions are ignored. Calls to rollback a transaction cause the driver to throw an exception indicating that no transaction is started. Metadata indicates that the driver supports transactions and the ReadUncommitted transaction isolation level.

If set to `3` (Full), the driver passes explicit transaction control commands such as `BEGIN TRANSACTION`, `COMMIT`, and `ROLLBACK` to the service.

Default Value

`3` (Full)

Truststore

Attribute

Truststore (ts)

Purpose

Specifies the directory of the truststore file to be used when 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

`string`

where:

`string`

is the directory of the truststore file.

Notes

- This value overrides the directory of the truststore file that is specified by the javax.net.ssl.trustStore Java system property. If this property is not specified, the truststore directory is specified by the javax.net.ssl.trustStore java system property.
- This option is ignored if ValidServerCertificate=`0`.

Default Value

No default value

See also

TLS/SSL encryption on page 79

Truststore Password

Attribute

TruststorePassword (tsp)

Valid Values

`string`

where:

`string`

is a valid password for the truststore file.

Behavior

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

Notes

- This value overrides the directory of the truststore file that is specified by the javax.net.ssl.trustStore Java system property. If this property is not specified, the truststore directory is specified by the javax.net.ssl.trustStore java system property.
- This option is ignored if ValidServerCertificate=`0`.

Default Value

No default value

See also

TLS/SSL encryption on page 79

Uppercase Identifiers

Attribute

UppercaseIdentifiers (uci)

Purpose

Specifies whether the driver maps all identifier names to uppercase. By default, the driver maps all identifier names to uppercase.

Valid Values

`0` | `1`

Behavior

If set to `1` (true), the driver maps identifiers to uppercase.

If set to `0` (false), the driver maps identifiers to the mixed case name of the object being mapped. If mixed case identifiers are used, SQL statements must enclose those identifiers in double quotes and the case of the identifier must exactly match the case of the identifier name. In addition, object names in results returned from catalog functions are returned in the case that they are stored in the database.

Example

If `UppercaseIdentifiers=false`, to query the Account table you specify:

```
SELECT "id", "name" FROM "Account"
```

Notes

- Do not change the value of Uppercase Identifiers unless the data source you are connecting to has objects with names that differ only by case.

- The setting for this option is written to the configuration file when generating the schema map. If you attempt a subsequent connection using the configuration file and specify a different value for this option, the driver will return an error.

Default Value

`1` (true)

See also

Using identifiers on page 96

User

Attribute

User

Purpose

Specifies the user ID for user ID/password authentication.

Valid Values

`userid`

where:

`userid`

is a valid user ID with permissions to access the database using user ID/password authentication.

Default Value

No default value

See also

Authentication on page 75

Validate Server Certificate

Attribute

ValidateServerCertificate (vsc)

Purpose

Determines whether the driver validates the certificate that is sent by the MongoDB server when SSL encryption is enabled. When using SSL server authentication, any certificate that is sent by the server must be issued by a trusted Certificate Authority (CA).

Valid Values

`0` | `1`

Behavior

If set to `1` (true), the driver validates the certificate that is sent by the server. Any certificate from the server must be issued by a trusted CA in the truststore file. If the Host Name In Certificate (HostNameInCertificate) option is specified, the driver also validates the certificate using a host name. The Host Name In Certificate option is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

If set to `0` (false), the driver does not validate the certificate that is sent by the server. The driver ignores any truststore information that is specified by the Truststore (TrustStore) and Truststore Password (TrustStorePassword) options or Java system properties.

Notes

- Truststore information is specified using the Truststore and Truststore Password options or by using Java system properties.
- Allowing the driver to trust any certificate that is returned from the server even if the issuer is not a trusted CA is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment.

Default Value

`1` (true)

See also

TLS/SSL encryption on page 79

Varchar Threshold

Attribute

VarcharThreshold (vth)

Purpose

Specifies the threshold at which the driver describes character columns as type LONGVARCHAR. This option allows you to fetch columns that would otherwise exceed the upper limit of the VARCHAR type for some third-party applications, such as SQL Server Linked Server.

Valid Values

x

where:

x

is the maximum size in characters of columns the driver will describe as VARCHAR.

Notes

- For STRING columns, the driver determines the column size to be 50% greater than the longest sampled value in the column. This behavior allows for values larger than those identified through sampling. If the adjusted column size exceeds the value specified for this option, the driver describes the column as LONGVARCHAR when calling SQLDescribeCol and SQLColumns.

Default Value

`4000`

7

Supported SQL statements and extensions

The MongoDB driver provides support for the SQL statements and the SQL extensions described in this chapter. SQL extensions are denoted by an (EXT) in the topic title.

For details, see the following topics:

- Create View
- Delete
- Drop View
- Insert
- Refresh Map (EXT)
- Reload Map (EXT)
- Select
- Update
- SQL Expressions
- Subqueries
- Custom function escapes

Create View

Purpose

The Create View statement creates a new local view. A view is analogous to a named query. The view's query can refer to any combination of remote and local tables as well as other views. Views are read-only; they cannot be updated.

Note: This statement does not affect native MongoDB views.

Syntax

```
CREATE VIEW view_name[(view_column,...)] AS
SELECT ... FROM ... [WHERE Expression]
  [ORDER BY order_expression [, ...]]
  [LIMIT limit [OFFSET offset]];
```

where:

view_name

specifies the name of the view.

view_column

specifies the column associated with the view. Multiple column names must be separated by commas.

The other commands used for Create View are the same as those used for Select (see "Select").

Notes

- A view can be thought of as a virtual table. A Select statement is stored in the database; however, the data accessible through a view is not stored in the database. The result set of the Select statement forms the virtual table returned by the view. You can use this virtual table by referring to the view name in SQL statements the same way you refer to a table. A view is used to perform any or all of these functions:
  - Restrict a user to specific rows in a table.
  - Restrict a user to specific columns.
  - Join columns from multiple tables so that they function like a single table.
  - Aggregate information instead of supplying details. For example, the sum of a column, or the maximum or minimum value from a column can be presented.
- Views are created by defining the Select statement that retrieves the data to be presented by the view.
- The Select statement in a View definition must return columns with distinct names. If the names of two columns in the Select statement are the same, use a column alias to distinguish between them. Alternatively, you can define a list of new columns for a view.

Example A

This example creates a view named myOpportunities that selects data from three database tables to present a virtual table of data.

```
CREATE VIEW myOpportunities AS
SELECT a.name AS AccountName,
       o.name AS OpportunityName,
       o.amount AS Amount,
       o.description AS Description
FROM Opportunity o INNER JOIN Account a
       ON o.AccountId = a.id
       INNER JOIN User u
       ON o.OwnerId = u.id
WHERE u.name = 'MyName'
       AND o.isClosed = 'false'
ORDER BY Amount desc
```

You can then refer to the myOpportunities view in statements just as you would refer to a table. For example:

```
SELECT * FROM myOpportunities;
```

Example B

The myOpportunities view contains a detailed description for each opportunity, which may not be needed when only a summary is required. A view can be built that selects only specific myOpportunities columns as shown in the following example:

```
CREATE VIEW myOpps_NoDesc as
SELECT AccountName,
       OpportunityName,
       Amount
FROM myOpportunities
```

The view selects the name column from both the opportunity and account tables. These columns are assigned the alias OpportunityName and AccountName, respectively.

See also

Select on page 167
Local views on page 100

Delete

Purpose

The Delete statement is used to delete rows from a table.

Syntax

`DELETE FROM` *`table_name`* `[WHERE` *`search_condition`*`]`

where:

`table_name`

specifies the name of the table from which you want to delete rows.

search_condition

is an expression that identifies which rows to delete from the table.

Notes

- The Where clause determines which rows are to be deleted. Without a Where clause, all rows of the table are deleted, but the table is left intact. See "Where Clause" for information about the syntax of Where clauses. Where clauses can contain subqueries.

Example A

This example shows a Delete statement on the emp table.

```
DELETE FROM emp WHERE emp_id = 'E10001'
```

Each Delete statement removes every record that meets the conditions in the Where clause. In this case, every record having the employee ID `E10001` is deleted. Because employee IDs are unique in the employee table, at most, one record is deleted.

Example B

This example shows using a subquery in a Delete clause.

```
DELETE FROM emp WHERE dept_id = (SELECT dept_id FROM dept WHERE dept_name =
'Marketing')
```

The records of all employees who belong to the department named Marketing are deleted.

Notes

- Insert, Update, and Delete are supported for MongoDB for simple types at the root level.
- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.

See also

Where Clause on page 173
Where Clause on page 173

Drop View

Purpose

The Drop View statement drops a local view.

Note: This statement does not affect native MongoDB views.

Syntax

```
DROP VIEW view_name [IF EXISTS] [RESTRICT | CASCADE]
```

where:

view_name

specifies the name of a view.

`IF EXISTS`

specifies that an error is not to be returned if the view does not exist.

`RESTRICT`

is in effect by default, meaning that the drop fails if any other view refers to this view.

`CASCADE`

silently drops all dependent local views.

See also

Local views on page 100

Insert

Purpose

The Insert statement is used to add new rows to a local table. You can specify either of the following options:

- List of values to be inserted as a new row
- Select statement that copies data from another table to be inserted as a set of new rows

Syntax

```
INSERT INTO table_name [(column_name[,column_name]...)] {VALUES (expression
[,expression]...) | select_statement}
```

table_name

is the name of the table in which you want to insert rows.

column_name

is optional and specifies an existing column. Multiple column names (a column list) must be separated by commas. A column list provides the name and order of the columns, the values of which are specified in the Values clause. If you omit a *column_name* or a column list, the value expressions must provide values for all columns defined in the table and must be in the same order that the columns are defined for the table. Table columns that do not appear in the column list are populated with the default value, or with NULL if no default value is specified.

expression

is the list of expressions that provides the values for the columns of the new record. Typically, the expressions are constant values for the columns. Character string values must be enclosed in single quotation marks ('). See "Literals" for more information.

select_statement

is a query that returns values for each *column_name* value specified in the column list. Using a Select statement instead of a list of value expressions lets you select a set of rows from one table and insert it into another table using a single Insert statement. The Select statement is evaluated before any values are inserted. This query cannot be made on the table into which values are inserted. See "Select" for information about Select statements.

Notes

- Insert, Update, and Delete are supported for MongoDB for simple types at the root level.
- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.

See also

Literals on page 180
Select on page 167
Literals on page 180
Select on page 167

Refresh Map (EXT)

Purpose

The REFRESH MAP statement adds newly discovered objects to your relational view of native data. It also incorporates any configuration changes made to your relational view by reloading the schema map and associated files.

Syntax

```
REFRESH MAP [NEW]
```

where

`NEW`

limits the sampling performed by the driver to only those collections that are newly discovered. When executing REFRESH MAP, this can offer significant performance gains if you only want to sample new collections or the existing collections are unchanged. If NEW is not specified in the statement, all collections are resampled.

Notes

- Newly discovered objects are mapped using the same relational view specified by the SchemaFormat property during your initial connection.
- REFRESH MAP is an expensive query since it involves the discovery of native data. However, by specifying NEW in the statement, you can reduce the overhead associated with this query by sampling only new collections.

Reload Map (EXT)

Purpose

The RELOAD MAP statement reloads the schema map and associated files. This statement allows you to update your relational view of native data while the driver is connected to the data store.

Syntax

```
RELOAD MAP
```

Notes

- RELOAD MAP does not discover changes made to the native data store.

Select

Purpose

Use the Select statement to fetch results from one or more tables.

Syntax

```
SELECT select_clause from_clause
[where_clause]
[groupby_clause]
[having_clause]
[{UNION [ALL | DISTINCT] |
 {MINUS [DISTINCT] | EXCEPT [DISTINCT]} |
 INTERSECT [DISTINCT]} select_statement]
[limit_clause]
```

where:

select_clause

specifies the columns from which results are to be returned by the query. See "Select Clause" for a complete explanation.

from_clause

specifies one or more tables on which the other clauses in the query operate. See "From Clause" for a complete explanation.

where_clause

is optional and restricts the results that are returned by the query. See "Where Clause" for a complete explanation.

groupby_clause

is optional and allows query results to be aggregated in terms of groups. See "Group By Clause" for a complete explanation.

having_clause

is optional and specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than $200,000). See "Having Clause" for a complete explanation.

`UNION`

is an optional operator that combines the results of the left and right Select statements into a single result. See "Union Operator" for a complete explanation.

`INTERSECT`

is an optional operator that returns a single result by keeping any distinct values from the results of the left and right Select statements. See "Intersect Operator" for a complete explanation.

`EXCEPT | MINUS`

are synonymous optional operators that returns a single result by taking the results of the left Select statement and removing the results of the right Select statement. See "Except and Minus Operators" for a complete explanation.

orderby_clause

is optional and sorts the results that are returned by the query. See "Order By Clause" for a complete explanation.

limit_clause

is optional and places an upper bound on the number of rows returned in the result. See "Limit Clause" for a complete explanation.

See also

Select Clause on page 169
From Clause on page 171
Where Clause on page 173
Group By Clause on page 174
Having Clause on page 174
Union Operator on page 175
Intersect Operator on page 176
Except and Minus Operators on page 177
Order By Clause on page 177
Limit Clause on page 178
Select Clause on page 169
From Clause on page 171
Where Clause on page 173
Group By Clause on page 174
Having Clause on page 174
Union Operator on page 175

Intersect Operator on page 176
Except and Minus Operators on page 177
Order By Clause on page 177
Limit Clause on page 178

Select Clause

Purpose

Use the Select clause to specify with a list of column expressions that identify columns of values that you want to retrieve or an asterisk (*) to retrieve the value of all columns.

Syntax

```
SELECT [{LIMIT offsetnumber | TOP number}] [ALL | DISTINCT] {* | column_expression
[[AS] column_alias] [,column_expression [[AS] column_alias], ...]}
```

where:

`LIMIT offset number`

creates the result set for the Select statement first and then discards the first number of rows specified by *offset* and returns the number of remaining rows specified by *number*. To not discard any of the rows, specify 0 for *offset*, for example, `LIMIT 0 number`. To discard the first *offset* number of rows and return all the remaining rows, specify 0 for *number*, for example, `LIMIT offset0`.

`TOP number`

is equivalent to `LIMIT 0number`.

column_expression

can be simply a column name (for example, `last_name`). More complex expressions may include mathematical operations or string manipulation (for example, `salary * 1.05`). See "SQL Expressions" for details. *column_expression* can also include aggregate functions. See "Aggregate Functions" for details.

column_alias

can be used to give the column a descriptive name. For example, to assign the alias department to the column dep:

```
SELECT dep AS department FROM emp
```

`DISTINCT`

eliminates duplicate rows from the result of a query. This operator can precede the first column expression. For example:

```
SELECT DISTINCT dep FROM emp
```

Notes

- Separate multiple column expressions with commas (for example, `SELECT last_name, first_name, hire_date`).

- Column names can be prefixed with the table name or table alias. For example, `SELECT emp.last_name or e.last_name`, where `e` is the alias for the table `emp`.
- NULL values are not treated as distinct from each other. The default behavior is that all result rows be returned, which can be made explicit with the keyword `ALL`.

See also

SQL Expressions on page 180
Aggregate Functions on page 170
SQL Expressions on page 180
Aggregate Functions on page 170

Aggregate Functions

Aggregate functions can also be a part of a Select clause. Aggregate functions return a single value from a set of rows. An aggregate can be used with a column name (for example, `AVG(salary)`) or in combination with a more complex column expression (for example, `AVG(salary * 1.07)`). The column expression can be preceded by the `DISTINCT` operator. The `DISTINCT` operator eliminates duplicate values from an aggregate expression.

The following table lists supported aggregate functions.

Table 38: Aggregate Functions

Aggregate	Returns
AVG	The average of the values in a numeric column expression. For example, `AVG(salary)` returns the average of all salary column values.
COUNT	The number of values in any field expression. For example, `COUNT(name)` returns the number of name values. When using `COUNT` with a field name, `COUNT` returns the number of non-NULL column values. A special example is `COUNT(*)`, which returns the number of rows in the set, including rows with NULL values.
MAX	The maximum value in any column expression. For example, `MAX(salary)` returns the maximum salary column value.
MIN	The minimum value in any column expression. For example, `MIN(salary)` returns the minimum salary column value.
SUM	The total of the values in a numeric column expression. For example, `SUM(salary)` returns the sum of all salary column values.

Example A

In the following example, only distinct last name values are counted. The default behavior is that all duplicate values be returned, which can be made explicit with `ALL`.

```
COUNT (DISTINCT last_name)
```

Example B

The following example uses the `COUNT`, `MAX`, and `AVG` aggregate functions:

```
SELECT
     COUNT(amount) AS numOpportunities,
     MAX(amount) AS maxAmount,
     AVG(amount) AS avgAmount
FROM opportunity o INNER JOIN user u
     ON o.ownerId = u.id
WHERE o.isClosed = 'false' AND
     u.name = 'MyName'
```

From Clause

Purpose

The From clause indicates the tables to be used in the Select statement.

Syntax

`FROM` *`table_name`* `[`*`table_alias`*`] [,...]`

where:

table_name

is the name of a table or a subquery. Multiple tables define an implicit inner join among those tables. Multiple table names must be separated by a comma. For example:

```
SELECT * FROM emp, dep
```

Subqueries can be used instead of table names. Subqueries must be enclosed in parentheses. See "Subquery in a From Clause" for an example.

table_alias

is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias.

Example

The following example specifies two table aliases, `e` for `emp` and `d` for `dep`:

```
SELECT e.name, d.deptName
FROM emp e, dep d
WHERE e.deptId = d.id
```

table_alias is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias. For example, given the table specification:

```
FROM emp E
```

you may refer to the last_name field as E.last_name. Table aliases must be used if the Select statement joins a table to itself. For example:

```
SELECT * FROM emp E, emp F WHERE E.mgr_id = F.emp_id
```

The equal sign (=) includes only matching rows in the results.

See also

Subquery in a From Clause on page 173

Outer Join Escape Sequences

Purpose

The SQL-92 left, right, and full outer join syntax is supported.

Syntax

```
{oj outer-join}
```

where *outer-join* is

```
table-reference {LEFT | RIGHT | FULL} OUTER JOIN {table-reference | outer-join} ON
search-condition
```

where *table-reference* is a database table name, and *search-condition* is the join condition you want to use for the tables.

```
Example: SELECT Customers.CustID, Customers.Name, Orders.OrderID, Orders.Status FROM {oj
 Customers LEFT OUTER JOIN Orders ON Customers.CustID=Orders.CustID} WHERE
Orders.Status='OPEN'
```

The following outer join escape sequences are supported by MongoDB databases:

- Left outer joins
- Right outer joins
- Full outer joins
- Nested outer joins

Join in a From Clause

Purpose

You can use a Join as a way to associate multiple tables within a Select statement. Joins may be either explicit or implicit. For example, the following is the example from the previous section restated as an explicit inner join:

```
SELECT * FROM emp INNER JOIN dep ON id=empId
SELECT e.name, d.deptName
FROM emp e INNER JOIN dep d ON e.deptId = d.id;
```

whereas the following is the same statement as an implicit inner join:

```
SELECT * FROM emp, dep WHERE emp.deptID=dep.id
```

Syntax

```
FROM table_name {RIGHT OUTER | INNER | LEFT OUTER | CROSS | FULL OUTER} JOIN table.key
 ON search-condition
```

Example

In this example, two tables are joined using `LEFT OUTER JOIN`. `T1`, the first table named includes nonmatching rows.

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.key = T2.key
```

If you use a `CROSS JOIN`, no `ON` expression is allowed for the join.

Subquery in a From Clause

Subqueries can be used in the From clause in place of table references (*table_name*).

Example

```
SELECT * FROM (SELECT * FROM emp WHERE sal > 10000) new_emp, dept WHERE
new_emp.deptno = dept.deptno
```

See also

Subqueries on page 187
Subqueries on page 187

Where Clause

Purpose

Specifies the conditions that rows must meet to be retrieved.

Syntax

`WHERE` *expr1rel_operatorexpr2*

where:

expr1

is either a column name, literal, or expression.

expr2

is either a column name, literal, expression, or subquery. Subqueries must be enclosed in parentheses.

rel_operator

is the relational operator that links the two expressions.

Example

The following Select statement retrieves the first and last names of employees that make at least $20,000.

```
SELECT last_name, first_name FROM emp WHERE salary >= 20000
```

See also

Subqueries on page 187
SQL Expressions on page 180
Subqueries on page 187

SQL Expressions on page 180

Group By Clause

Purpose

Specifies the names of one or more columns by which the returned values are grouped. This clause is used to return a set of aggregate values.

Syntax

```
GROUP BY column_expression [,...]
```

where:

column_expression

is either a column name or a SQL expression. Multiple values must be separated by a comma. If *column_expression* is a column name, it must match one of the column names specified in the Select clause. Also, the Group By clause must include all non-aggregate columns specified in the Select list.

Example

The following example totals the salaries in each department:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id
```

This statement returns one row for each distinct department ID. Each row contains the department ID and the sum of the salaries of the employees in the department.

Notes

The driver uses the MongoDB aggregation framework whenever possible. In some instances, MongoDB may aggregate data in unexpected ways. For example, if you use the GROUP BY clause to return zip codes and the database contains the zip code 15237 as both a string and an integer, then two rows will be returned for 15237 (one for the string representation and another for the integer representation).

See also

Subqueries on page 187
SQL Expressions on page 180
Subqueries on page 187
SQL Expressions on page 180

Having Clause

Purpose

Specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than $200,000). This clause is valid only if you have already defined a Group By clause.

Syntax

```
HAVING expr1 rel_operator expr2
```

where:

expr1 | *expr2*

can be column names, constant values, or expressions. These expressions do not have to match a column expression in the Select clause. See "SQL Expressions" for details regarding SQL expressions.

rel_operator

is the relational operator that links the two expressions.

Example

The following example returns only the departments that have salaries totaling more than $200,000:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id HAVING sum(salary) > 200000
```

Notes

The driver uses the MongoDB aggregation framework whenever possible. In some instances, MongoDB may aggregate data in unexpected ways. For example, if you use the HAVING clause to return zip codes and the database contains the zip code 15237 as both a string and an integer, then two rows will be returned for 15237 (one for the string representation and another for the integer representation).

See also

Subqueries on page 187
SQL Expressions on page 180
Subqueries on page 187
SQL Expressions on page 180

Union Operator

Purpose

Combines the results of two Select statements into a single result. The single result is all the returned rows from both Select statements. By default, duplicate rows are not returned. To return duplicate rows, use the All keyword (`UNION ALL`).

Syntax

```
select_statement
UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT]} | INTERSECT
[DISTINCT]select_statement
```

Notes

- When using the Union operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
UNION
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
UNION
SELECT salary, last_name FROM raises
```

Intersect Operator

Purpose

Intersect operator returns a single result set. The result set contains rows that are returned by both Select statements. Duplicates are returned unless the Distinct operator is added.

Syntax

```
select_statement
INTERSECT [DISTINCT]
select_statement
```

where:

`DISTINCT`

eliminates duplicate rows from the results.

Notes

- When using the Intersect operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
INTERSECT [DISTINCT]
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
INTERSECT
SELECT salary, last_name FROM raises
```

Except and Minus Operators

Purpose

Return the rows from the left Select statement that are not included in the result of the right Select statement.

Syntax

```
select_statement
{EXCEPT [DISTINCT] | MINUS [DISTINCT]}
select_statement
```

where:

`DISTINCT`

eliminates duplicate rows from the results.

Notes

- When using one of these operators, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
EXCEPT
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
EXCEPT
SELECT salary, last_name FROM raises
```

Order By Clause

Purpose

The Order By clause specifies how the rows are to be sorted.

Syntax

```
ORDER BY sort_expression [DESC | ASC] [,...]
```

where:

sort_expression

is either the name of a column, a column alias, a SQL expression, or the positioned number of the column or expression in the select list to use.

The default is to perform an ascending (ASC) sort.

Example

To sort by last_name and then by first_name, you could use either of the following Select statements:

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY last_name, first_name
```

or

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY 2,3
```

In the second example, last_name is the second item in the Select list, so ORDER BY 2,3 sorts by last_name and then by first_name.

See also

SQL Expressions on page 180
SQL Expressions on page 180

Limit Clause

Purpose

Places an upper bound on the number of rows returned in the result.

Syntax

LIMIT *number_of_rows* [OFFSET *offset_number*]

where:

number_of_rows

specifies a maximum number of rows in the result. A negative number indicates no upper bound.

OFFSET

specifies how many rows to skip at the beginning of the result set. *offset_number* is the number of rows to skip.

Notes

- In a compound query, the Limit clause can appear only on the final Select statement. The limit is applied to the entire query, not to the individual Select statement to which it is attached.

Example

The following example returns a maximum of 20 rows.

```
SELECT last_name, first_name FROM emp WHERE salary > 20000 ORDER BY dept_idc LIMIT
20
```

Update

Purpose

An Update statement changes the value of columns in the selected rows of a table.

Syntax

UPDATE *table_name* SET *column_name* = *expression*

[, *column_name* = *expression*] [WHERE *conditions*]

table_name

is the name of the table for which you want to update values.

column_name

is the name of a column, the value of which is to be changed. Multiple column values can be changed in a single statement.

expression

is the new value for the column. The expression can be a constant value or a subquery that returns a single value. Subqueries must be enclosed in parentheses.

Example A

The following example changes every record that meets the conditions in the Where clause. In this case, the salary and exempt status are changed for all employees having the employee ID `E10001`. Because employee IDs are unique in the emp table, only one record is updated.

```
UPDATE emp SET salary=32000, exempt=1

WHERE emp_id = 'E10001'
```

Example B

The following example uses a subquery. In this example, the salary is changed to the average salary in the company for the employee having employee ID `E10001`.

```
UPDATE emp SET salary = (SELECT avg(salary) FROM emp)

WHERE emp_id = 'E10001'
```

Notes

- Insert, Update, and Delete are supported for MongoDB for simple types at the root level.
- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.
- A Where clause can be used to restrict which rows are updated.

See also

Subqueries on page 187
Where Clause on page 173
Subqueries on page 187
Where Clause on page 173

SQL Expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. You can use expressions in the Where, and Having of Select statements; and in the Set clauses of Update statements.

Expressions enable you to use mathematical operations as well as character string manipulation operators to form complex queries.

The MongoDB driver supports both unquoted and quoted identifiers. An unquoted identifier must start with an ASCII alpha character and can be followed by zero or more ASCII alphanumeric characters. Unquoted identifiers are converted to uppercase before being used.

Quoted identifiers must be enclosed in double quotation marks (""). A quoted identifier can contain any Unicode character including the space character. The MongoDB driver recognizes the Unicode escape sequence \uxxxx as a Unicode character. You can specify a double quotation mark in a quoted identifier by escaping it with a double quotation mark.

The maximum length of both quoted and unquoted identifiers is 128 characters.

Valid expression elements are:

- Column names
- Literals
- Operators
- Functions

Column Names

The most common expression is a simple column name. You can combine a column name with other expression elements.

Literals

Literals are fixed data values. For example, in the expression `PRICE * 1.05`, the value 1.05 is a constant. Literals are classified into types, including the following:

- Binary
- Character string
- Date
- Floating point
- Integer

- Numeric
- Time
- Timestamp

The following table describes the literal format for supported SQL data types.

Table 39: Literal Syntax Examples

SQL Type	Literal Syntax	Example
BIGINT	*n* where *n* is any valid integer value in the range of the BIGINT data type	`12 or -34 or 0`
BOOLEAN	Min Value: 0 Max Value: 1	`0` `1`
DATE	DATE'*date*'	`'2010-05-21'`
DATETIME	TIMESTAMP'*ts*'	`'2010-05-21 18:33:05.025'`

DECIMAL	*n.f*
	where:
	n
	is the integral part
	f
	is the fractional part
		`0.25`
		`3.1415`
		`-7.48`

DOUBLE	*n.f*Ex where: *n* is the integral part *f* is the fractional part *x* is the exponent	`1.2E0` or `2.5E40` or `-3.45E2` or `5.67E-4`
INTEGER	*n* where *n* is a valid integer value in the range of the INTEGER data type	`12` or `-34` or `0`
LONGVARBINARY	'*hex_value*'	`'000482ff'`
LONGVARCHAR	'*value*'	`'This is a string literal'`

SQL Type	Literal Syntax	Example
TIME	TIME'*time*'	`'2010-05-21 18:33:05.025'`
VARCHAR	'*value*'	`'This is a string literal'`

Character String Literals

Text specifies a character string literal. A character string literal must be enclosed in single quotation marks. To represent one single quotation mark within a literal, you must enter two single quotation marks. When the data in the fields is returned to the client, trailing blanks are stripped.

A character string literal can have a maximum length of 32 KB, that is, (32*1024) bytes.

Example

```
'Hello'
'Jim''s friend is Joe'
```

Numeric Literals

Unquoted numeric values are treated as numeric literals. If the unquoted numeric value contains a decimal point or exponent, it is treated as a real literal; otherwise, it is treated as an integer literal.

Example

```
+1894.1204
```

Binary Literals

Binary literals are represented with single quotation marks. The valid characters in a binary literal are 0-9, a-f, and A-F.

Example

```
'00af123d'
```

Date/Time Literals

Date and time literal values are enclosed in single quotion marks ('*value*').

- The format for a Date literal is DATE'*date*'.
- The format for a Time literal is TIME'*time*'.
- The format for a Timestamp literal is TIMESTAMP'*ts*'.

Integer Literals

Integer literals are represented by a string of numbers that are not enclosed in quotation marks and do not contain decimal points.

Notes

- Integer constants must be whole numbers; they cannot contain decimals.
- Integer literals can start with sign characters (+/-).

Example

```
1994 or -2
```

Operators

This section describes the operators that can be used in SQL expressions.

Unary Operator

A unary operator operates on only one operand.

operator operand

Binary Operator

A binary operator operates on two operands.

operand1 operator operand2

If an operator is given a null operand, the result is always null. The only operator that does not follow this rule is concatenation (||).

Arithmetic Operators

You can use an arithmetic operator in an expression to negate, add, subtract, multiply, and divide numeric values. The result of this operation is also a numeric value. The + and - operators are also supported in date/time fields to allow date arithmetic. The following table lists the supported arithmetic operators.

Table 40: Arithmetic Operators

Operator	Purpose	Example
+ -	Denotes a positive or negative expression. These are unary operators.	`SELECT * FROM emp WHERE comm = -1`
* /	Multiplies, divides. These are binary operators.	`UPDATE emp SET sal = sal + sal * 0.10`
+ -	Adds, subtracts. These are binary operators.	`SELECT sal + comm FROM emp WHERE empno > 100`

Concatenation Operator

The concatenation operator manipulates character strings. The following table lists the only supported concatenation operator.

Table 41: Concatenation Operator

<table>
<tr><th>Operator</th><th>Purpose</th><th>Example</th></tr>
<tr><td>||</td><td>Concatenates character strings.</td><td>SELECT 'Name is' || ename FROM emp</td></tr>
</table>

The result of concatenating two character strings is the data type VARCHAR.

Comparison Operators

Comparison operators compare one expression to another. The result of such a comparison can be TRUE, FALSE, or UNKNOWN (if one of the operands is NULL). The MongoDB driver considers the UNKNOWN result as FALSE.

The following table lists the supported comparison operators.

Table 42: Comparison Operators

Operator	Purpose	Example
=	Equality test.	SELECT * FROM emp WHERE sal = 1500
!=<>	Inequality test.	SELECT * FROM emp WHERE sal != 1500
><	“Greater than" and "less than" tests.	SELECT * FROM emp WHERE sal > 1500 SELECT * FROM emp WHERE sal < 1500
>=<=	“Greater than or equal to" and "less than or equal to" tests.	SELECT * FROM emp WHERE sal >= 1500 SELECT * FROM emp WHERE sal <= 1500
ESCAPE clause in LIKE operator LIKE ’pattern string’ ESCAPE ’c’	The Escape clause is supported in the LIKE predicate to indicate the escape character. Escape characters are used in the pattern string to indicate that any wildcard character that is after the escape character in the pattern string should be treated as a regular character. The default escape character is backslash (\).	SELECT * FROM emp WHERE ENAME LIKE 'J%_%' ESCAPE '\' This matches all records with names that start with letter 'J' and have the '_' character in them. SELECT * FROM emp WHERE ENAME LIKE 'JOE_JOHN' ESCAPE '\' This matches only records with name ’JOE_JOHN’.
[NOT] IN	“Equal to any member of" test.	SELECT * FROM emp WHERE job IN ('CLERK','ANALYST') SELECT * FROM emp WHERE sal IN (SELECT sal FROM emp WHERE deptno = 30)
[NOT] BETWEEN x AND y	"Greater than or equal to x" and "less than or equal to y."	SELECT * FROM emp WHERE sal BETWEEN 2000 AND 3000

Operator	Purpose	Example
EXISTS	Tests for existence of rows in a subquery.	`SELECT empno, ename, deptno FROM emp e WHERE EXISTS (SELECT deptno FROM dept WHERE e.deptno = dept.deptno)`
IS [NOT] NULL	Tests whether the value of the column or expression is NULL.	`SELECT * FROM emp WHERE ename IS NOT NULL SELECT * FROM emp WHERE ename IS NULL`

Logical Operators

A logical operator combines the results of two component conditions to produce a single result or to invert the result of a single condition. The following table lists the supported logical operators.

Table 43: Logical Operators

Operator	Purpose	Example
NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN.	`SELECT * FROM emp WHERE NOT (job IS NULL)` `SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000)`
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE; otherwise, returns UNKNOWN.	`SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10`
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE; otherwise, returns UNKNOWN.	`SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10`

Example

In the Where clause of the following Select statement, the `AND` logical operator is used to ensure that managers earning more than $1000 a month are returned in the result:

```
SELECT * FROM emp WHERE jobtitle = manager AND sal > 1000
```

Operator Precedence

As expressions become more complex, the order in which the expressions are evaluated becomes important. The following table shows the order in which the operators are evaluated. The operators in the first line are evaluated first, then those in the second line, and so on. Operators in the same line are evaluated left to right in the expression.You can change the order of precedence by using parentheses. Enclosing expressions in parentheses forces them to be evaluated together.

Table 44: Operator Precedence

Precedence	Operator
1	+ (Positive), - (Negative)
2	*(Multiply), / (Division)
3	+ (Add), - (Subtract)
4	|| (Concatenate)
5	=, >, <, >=, <=, <>, != (Comparison operators)
6	NOT, IN, LIKE
7	AND
8	OR

Example A

The query in the following example returns employee records for which the department number is 1 or 2 and the salary is greater than $1000:

```
SELECT * FROM emp WHERE (deptno = 1 OR deptno = 2) AND sal > 1000
```

Because parenthetical expressions are forced to be evaluated first, the OR operation takes precedence over AND.

Example B

In the following example, the query returns records for all the employees in department 1, but only employees whose salary is greater than $1000 in department 2.

```
SELECT * FROM emp WHERE deptno = 1 OR deptno = 2 AND sal > 1000
```

The AND operator takes precedence over OR, so that the search condition in the example is equivalent to the expression `deptno = 1 OR (deptno = 2 AND sal > 1000).`

Functions

The MongoDB driver supports a number of functions that you can use in expressions, as listed and described in "Supported scalar functions."

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

Conditions

A condition specifies a combination of one or more expressions and logical operators that evaluates to either TRUE, FALSE, or UNKNOWN. You can use a condition in the Where clause of the Delete, Select, and Update statements; and in the Having clauses of Select statements. The following describes supported conditions.

Table 45: Conditions

Condition	Description
Simple comparison	Specifies a comparison with expressions or subquery results. `= , !=, <>, < , >, <=, <=`
Group comparison	Specifies a comparison with any or all members in a list or subquery. `[= , !=, <>, < , >, <=, <=] [ANY, ALL, SOME]`
Membership	Tests for membership in a list or subquery. `[NOT] IN`
Range	Tests for inclusion in a range. `[NOT] BETWEEN`
NULL	Tests for nulls. `IS NULL, IS NOT NULL`
EXISTS	Tests for existence of rows in a subquery. `[NOT] EXISTS`
LIKE	Specifies a test involving pattern matching. `[NOT] LIKE`
Compound	Specifies a combination of other conditions. `CONDITION [AND/OR] CONDITION`

Subqueries

A query is an operation that retrieves data from one or more tables or views. In this reference, a top-level query is called a Select statement, and a query nested within a Select statement is called a subquery.

A subquery is a query expression that appears in the body of another expression such as a Select, an Update, or a Delete statement. In the following example, the second Select statement is a subquery:

```
SELECT * FROM emp WHERE deptno IN (SELECT deptno FROM dept)
```

IN Predicate

Purpose

The In predicate specifies a set of values against which to compare a result set. If the values are being compared against a subquery, only a single column result set is returned.

Syntax

```
value [NOT] IN (value1, value2,...)

OR

value [NOT] IN (subquery)
```

Example

```
SELECT * FROM emp WHERE deptno IN

(SELECT deptno FROM dept WHERE dname <> 'Sales')
```

EXISTS Predicate

Purpose

The Exists predicate is true only if the cardinality of the subquery is greater than 0; otherwise, it is false.

Syntax

```
EXISTS (subquery)
```

Example

```
SELECT empno, ename, deptno FROM emp e WHERE EXISTS
(SELECT deptno FROM dept WHERE e.deptno = dept.deptno)
```

UNIQUE Predicate

Purpose

The Unique predicate is used to determine whether duplicate rows exist in a virtual table (one returned from a subquery).

Syntax

```
UNIQUE (subquery)
```

Example

```
SELECT * FROM dept d WHERE UNIQUE
(SELECT deptno FROM emp e WHERE e.deptno = d.deptno)
```

Correlated Subqueries

Purpose

A correlated subquery is a subquery that references a column from a table referred to in the parent statement. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a Select, Update, or Delete statement.

A correlated subquery answers a multiple-part question in which the answer depends on the value in each row processed by the parent statement. For example, you can use a correlated subquery to determine which employees earn more than the average salaries for their departments. In this case, the correlated subquery specifically computes the average salary for each department.

Syntax

```
SELECT select_list
   FROM table1 t_alias1
   WHERE expr rel_operator
   (SELECT column_list
       FROM table2 t_alias2
       WHERE t_alias1.columnrel_operatort_alias2.column)
UPDATE table1 t_alias1
   SET column =
   (SELECT expr
       FROM table2 t_alias2
        WHERE t_alias1.column = t_alias2.column)
DELETE FROM table1 t_alias1
   WHERE column rel_operator
   (SELECT expr
       FROM table2 t_alias2
       WHERE t_alias1.column = t_alias2.column)
```

Notes

- Correlated column names in correlated subqueries must be explicitly qualified with the table name of the parent.

Example A

The following statement returns data about employees whose salaries exceed their department average. This statement assigns an alias to `emp`, the table containing the salary information, and then uses the alias in a correlated subquery:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
  ORDER BY deptno
```

Example B

This is an example of a correlated subquery that returns row values:

```
SELECT * FROM dept "outer" WHERE 'manager' IN
  (SELECT managername FROM emp
  WHERE "outer".deptno = emp.deptno)
```

Example C

This is an example of finding the department number (`deptno`) with multiple employees:

```
SELECT * FROM dept main WHERE 1 <
  (SELECT COUNT(*) FROM emp WHERE deptno = main.deptno)
```

Example D

This is an example of correlating a table with itself:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
```

Custom function escapes

The MongoDB driver supports the custom function escape CAST_TO_NATIVE, as described in the following topic.

CAST_TO_NATIVE function escape

Purpose

The CAST_TO_NATIVE function escape can be used in a filter to select or insert a value of a specific native type.

In some cases, a value with a specific native type can be concealed or obscured because the driver describes a field with inconsistent native types as a column with a single SQL type. For example, the driver maps a MongoDB `_id` field with the native types String and ObjectId to a relational `_ID` column with the SQL_WVARCHAR type. In this context, the CAST_TO_NATIVE function escape allows users to send a value as it is defined in the MongoDB database, rather than how it is described in the relational model of the data.

Currently, CAST_TO_NATIVE can only be used with the ObjectID type in SELECT statement filters and literal INSERT values.

Syntax

```
{fn CAST_TO_NATIVE('value','native_type')}
```

where:

value

is the value to be selected or inserted.

native_type

is the native type that in part defines the value.

Example Select

The following statement selects records from `Account` where the `_ID` field has an ObjectID value of `507f1f77bcf86cd799439011`.

```
SELECT * FROM Account WHERE _ID = {fn
CAST_TO_NATIVE('507f1f77bcf86cd799439011','objectid')}
```

Example Insert

The following statement inserts the ObjectID value `507f1f77bcf86cd799439011` into the `_ID` field.

```
INSERT INTO Account(_ID) VALUES ({fn
CAST_TO_NATIVE('507f1f77bcf86cd799439011','objectid')})
```