insightsoftware

Simba MongoDB ODBC Data Connector

Installation and Configuration Guide

Version 2.5

November 2025

Contents

Copyright

Contact Us

www.insightsoftware.com

About This Guide

Purpose

The *Simba MongoDB ODBC Data Connector Installation and Configuration Guide* explains how to install and configure the Simba MongoDB ODBC Data Connector. The guide also provides details related to features of the connector.

Audience

The guide is intended for end users of the Simba MongoDB ODBC Connector, as well as administrators and developers integrating the connector.

Knowledge Prerequisites

To use the Simba MongoDB ODBC Connector, the following knowledge is helpful:

- Familiarity with the platform on which you are using the Simba MongoDB ODBC Connector
- Ability to use the data source to which the Simba MongoDB ODBC Connector is connecting
- An understanding of the role of ODBC technologies and driver managers in connecting to a data source
- Experience creating and configuring ODBC connections
- Exposure to SQL

Document Conventions

Italics is used when referring to book and document titles.

Bold is used in procedures for graphical user interface elements that a user clicks and text that a user types.

`Monospace font` indicates commands, source code, or contents of text files.

Note: A text box with a pencil icon indicates a short note appended to a paragraph.

Important: A text box with an exclamation mark indicates an important comment related to the preceding paragraph.

About the Simba MongoDB ODBC Connector

The Simba MongoDB ODBC Connector enables Business Intelligence (BI), analytics, and reporting on data that is stored in MongoDB databases. The connector complies with the ODBC 3.80 data standard and adds important functionality such as Unicode, as well as 32- and 64-bit support for high-performance computing environments on all platforms.

ODBC is one of the most established and widely supported APIs for connecting to and working with databases. At the heart of the technology is the ODBC connector, which connects an application to the database. For more information about ODBC, see: https://insightsoftware.com/blog/what-is-odbc/. For complete information about the ODBC specification, see the *ODBC API Reference* from the Microsoft documentation: https://docs.microsoft.com/en-us/sql/odbc/reference/syntax/odbc-api-reference.

The Simba MongoDB ODBC Connector is available for Microsoft® Windows®, Linux, AIX, and macOS platforms.

Note: The AIX connector is not available through the Simba website. To get this connector, contact the Sales & Solutions team:

- Phone number: +1.604.633.0008 ext 2
- Email: solutions@simba.com

The *Installation and Configuration Guide* is suitable for users who are looking to access MongoDB data from their desktop environment. Application developers might also find the information helpful. Refer to your application for details on connecting via ODBC.

Platform and Data source version support

The Simba MongoDB ODBC Connector supports Windows, macOS, and Linux operating systems. For detailed information on supported operating system and data source versions, please refer to the connector's release notes.

Windows Connector

This section provides an overview of the Connector in the Windows platform, outlining the required system specifications and the steps for installing and configuring the connector in Windows environments.

Windows System Requirements

Install the connector on client machines where the application is installed. Before installing the connector, make sure that you have the following:

- Administrator rights on your machine.
- 600 MB of available disk space
- Does not have version 1.8.4 or earlier of the Simba MongoDB ODBC Connector installed.

Before the connector can be used, the Visual C++ Redistributable for Visual Studio 2015-2022 with the same bitness as the connector must also be installed. If you obtained the connector from the Simba website, then your installation of the connector automatically includes this dependency. Otherwise, you must install the redistributable manually. You can download the installation packages for the redistributable at https://www.microsoft.com/en-ca/download/details.aspx?id=48145.

Installing the Connector in Windows

If you did not obtain this connector from the Simba website, you might need to follow a different installation procedure. For more information, see the *Simba OEM ODBC Connector Installation Guide*.

On 64-bit Windows operating systems, you can execute both 32-bit and 64-bit applications. However, 64-bit applications must use 64-bit connectors, and 32-bit applications must use 32-bit connectors. Make sure that you use a connector whose bitness matches the bitness of the client application:

- `SimbaMongoDBODBC32.msi` for 32-bit applications
- `SimbaMongoDBODBC64.msi` for 64-bit applications

You can install both versions of the connector on the same machine.

To install the Simba MongoDB ODBC Connector in Windows:

1. Depending on the bitness of your client application, double-click to run **Simba MongoDB <Version Number> 32-bit.msi** or **Simba MongoDB <Version Number> 64-bit.msi**.
2. Click **Next**.
3. Select the check box to accept the terms of the License Agreement if you agree, and then click **Next**.
4. To change the installation location, click **Change**, then browse to the desired folder, and then click **OK**. To accept the installation location, click **Next**.
5. Click **Install**.

6. When the installation completes, click **Finish**.
7. If you received a license file through email, then copy the license file into the `\lib` subfolder of the installation folder you selected above. You must have Administrator privileges when changing the contents of this folder.

Creating a Data Source Name in Windows

Typically, after installing the Simba MongoDB ODBC Connector, you need to create a Data Source Name (DSN).

Alternatively, for information about DSN-less connections, see Using a Connection String.

To create a Data Source Name in Windows:

1. From the Start menu, go to **ODBC Data Sources**.

   Note: Make sure to select the ODBC Data Source Administrator that has the same bitness as the client application that you are using to connect to MongoDB.

2. In the ODBC Data Source Administrator, click the **Drivers** tab, and then scroll down as needed to confirm that the Simba MongoDB ODBC Connectorappears in the alphabetical list of ODBC Drivers that are installed on your system.
3. Choose one:
   - To create a DSN that only the user currently logged into Windows can use, click the **User DSN** tab.
   - Or, to create a DSN that all users who log into Windows can use, click the **System DSN** tab.

     Note: It is recommended that you create a System DSN instead of a User DSN. Some applications load the data using a different user account, and might not be able to detect User DSNs that are created under another user account.

4. Click **Add**.
5. In the Create New Data Source dialog box, select and then click **Finish**. The DSN Setup dialog box opens.
6. In the **Data Source Name** field, type a name for your DSN.
7. Optionally, in the **Description** field, type relevant details about the DSN.
8. In the **Server** field, type the name or IP address of the host where your MongoDB instance is running.
9. In the **Port** field, type the number of the TCP port that the server uses to listen for client connections.

   Note:
   The default port used by MongoDB is 27017.

10. In the **Database** field, type the name of the database that you want to access.
11. If you are connecting to a replica set in your MongoDB implementation, select the **Connect to Replica Set** check box and then do the following:
    a. In the **Replica Set Name** field, type the name of the replica set (this is a required field).
    b. In the **Secondary Servers** field, type a comma-separated list of the servers in the replica set. You can indicate the TCP port that each server is using to listen for client connections by appending a colon (`:`) and the port number to the server name or IP address.
    c. In the **Read Preference** drop-down list, select the appropriate option to specify how the connector routes read operations to the members of a replica set.
12. If the database that you are connecting to requires authentication, then use the options in the Authentication area to configure authentication as needed. For more information, see Configuring Authentication in Windows.
13. To configure advanced connector options including write-back options, click **Advanced Options**. For more information, see Configuring Advanced Options in Windows.
14. To configure client-server verification over SSL, click **SSL Options**. For more information, see Configuring SSL Verification in Windows.
15. To launch the Schema Editor application and create or customize the schema definition that the connector uses when connecting to the database, click **Schema Editor**.

    Note:
    For information about how to use the Schema Editor, see the *Schema Editor User Guide* located in the installation directory of the connector.

    - in Windows 7 or earlier, the guide is available from the **Simba MongoDB ODBC Driver** program group in the Start menu.
    - in Windows 8 or later, you can search for the guide on the Start screen.

16. To configure logging behavior for the connector, click **Logging Options**. For more information, see Configuring Logging Options in Windows.
17. To test the connection, click **Test**. Review the results as needed, and then click **OK**.

    Note:
    If the connection fails, then confirm that the settings in the Simba MongoDB ODBC Driver DSN Setup dialog box are correct. Contact your MongoDB server administrator as needed.

18. To save your settings and close the Simba MongoDB ODBC Driver DSN Setup dialog box, click **OK**.
19. To close the ODBC Data Source Administrator, click **OK**.

Configuring Authentication in Windows

Some MongoDB databases require authentication. You can configure the Simba MongoDB ODBC Connector to provide your credentials and authenticate the connection to the database using one of the following methods:

- Using SCRAM-SHA-1
- Using SCRAM-SHA-256
- Using Kerberos
- Using LDAP

Note:
The MONGO-CR authentication mechanism is deprecated as of MongoDB version 3.0.

The Simba MongoDB ODBC Connector officially supports MongoDB 3.6 through 4.2 only, but still provides limited support for the MONGO-CR authentication mechanism. If authentication through SCRAM-SHA-1 fails, the connector automatically retries authentication using MONGO-CR instead, potentially enabling connections to MongoDB 2.x.

Using SCRAM-SHA-1

You can configure the connector to use the SCRAM-SHA-1 protocol to authenticate the connection. SCRAM-SHA-1 is the default authentication protocol used by MongoDB.

Note:
If authentication through SCRAM-SHA-1 fails, the connector automatically retries authentication using the MONGO-CR mechanism instead. MONGO-CR is deprecated as of MongoDB version 3.0.

To configure SCRAM-SHA-1 authentication in Windows:

1. To access authentication options, open the ODBC Data Source Administrator where you created the DSN, select the DSN, and then click **Configure**.
2. In the **Mechanism** drop-down list, select **MongoDB User Name and Password**.
3. To use a database other than the admin database to check your credentials, type the name of the database in the **Authentication Source** field.
4. In the **Username** field, type an appropriate user name for accessing the MongoDB database.
5. In the **Password** field, type the password corresponding to the user name you typed above.
6. To encrypt your credentials, select one of the following:
   - If the credentials are used only by the current Windows user, select **Current User Only**.
   - Or, if the credentials are used by all users on the current Windows machine, select **All Users Of This Machine**.

7. To save your settings and close the dialog box, click **OK**.

Using SCRAM-SHA-256

> **Note:**
>
> SCRAM-SHA-256 authentication is only supported on MongoDB version 4.0 and above.

You can configure the connector to use the SCRAM-SHA-256 protocol to authenticate the connection.

To configure SCRAM-SHA-256 authentication in Windows:

1. To access authentication options, open the ODBC Data Source Administrator where you created the DSN, select the DSN, and then click **Configure**.
2. In the **Mechanism** drop-down list, select **SCRAM-SHA-256**.
3. To use a database other than the admin database to check your credentials, type the name of the database in the **Authentication Source** field.
4. In the **Username** field, type an appropriate user name for accessing the MongoDB database.
5. In the **Password** field, type the password corresponding to the user name you typed above.
6. To encrypt your credentials, select one of the following:
   - If the credentials are used only by the current Windows user, select **Current User Only**.
   - Or, if the credentials are used by all users on the current Windows machine, select **All Users Of This Machine**.
7. To save your settings and close the dialog box, click **OK**.

Using Kerberos

You can configure the connector to use the Kerberos protocol to authenticate the connection.

Kerberos must be installed and configured before you can use this authentication mechanism. For information about how to install and configure Kerberos, see the MIT Kerberos Documentation: http://web.mit.edu/kerberos/krb5-latest/doc/.

To configure Kerberos authentication in Windows:

1. To access authentication options, open the ODBC Data Source Administrator where you created the DSN, select the DSN, and then click **Configure**.
2. In the **Mechanism** drop-down list, select **Kerberos**.
3. In the **Service Name** field, type the service name of the MongoDB server.
4. To save your settings and close the dialog box, click **OK**.

Using LDAP

You can configure the connector to use the LDAP protocol to authenticate the connection.

To configure LDAP authentication in Windows:

1. To access authentication options, open the ODBC Data Source Administrator where you created the DSN, select the DSN, and then click **Configure**.
2. In the **Mechanism** drop-down list, select **LDAP**.
3. In the **Username** field, type an appropriate user name for accessing the MongoDB database.
4. In the **Password** field, type the password corresponding to the user name you typed above.
5. To encrypt your credentials, select one of the following:
   - If the credentials are used only by the current Windows user, select **Current User Only**.
   - Or, if the credentials are used by all users on the current Windows machine, select **All Users Of This Machine**.
6. To save your settings and close the dialog box, click **OK**.

Configuring Advanced Options in Windows

You can configure advanced options to modify the behavior of the connector.

To configure advanced options in Windows:

1. To access advanced options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Advanced Options**.
2. To retrieve data using double-buffering instead of single-buffering, select the **Enable Double-Buffering** check box. You can configure the buffer size using the **Documents to fetch per block** field.
3. In the **Documents to fetch per block** field, type the maximum number of documents that a query returns at a time. This setting also determines the buffer size used when double-buffering is enabled.
4. To return MongoDB String data as SQL_WVARCHAR instead of SQL_VARCHAR, select the **Expose Strings as SQL_WVARCHAR** check box.
5. In the **String Column Size** field, type the maximum data length for String columns.
6. To return MongoDB Binary data as SQL_LONGVARBINARY instead of SQL_VARBINARY, select the **Expose Binary as SQL_LONGVARBINARY** check box.
7. In the **Binary Column Size** field, type the maximum data length for Binary columns.
8. To configure the connector to create a column for retrieving or storing documents as JSON-formatted strings:
   a. Select the **Enable JSON Read/Write Mode** check box.
   b. In the **JSON Column Size** field, type the default column length for the JSON fields.
9. To configure the connector to optimize joins between virtual tables and pass filtering and aggregation optimizations to the MongoDB database for handling, select the **Enable Passdown** check box.

10. To allow data values to pass filtering operations even if the values are stored as a different data type than the specified filter term, select the **Enable Mixed Type Filter** check box.

> **Note:**
> Mixed type filtering requires the connector to scan the entire MongoDB collection. You can disable this feature to increase the connector's performance, but note that doing so also alters the results of your queries.

11. To configure the connector to ignore null values during INSERT operations even when the values are explicitly provided in the statement, select the **Omit Explicit NULL Columns On Insert** check box.

12. To configure the connector to bypass schema validation, select the **Enable BypassDocumentValidation** check box.

13. To configure the connector to compress temporary tables for nested select queries, select the **Enable Temporary Table Compression** check box.

14. Use the options in the Metadata area to specify the schema definition to use when connecting to the database:

- To configure the connector to use a schema definition stored in a JSON file, select **Local File** from the **Mechanism** drop-down list, and then click **Browse** and select the JSON file that you want to use.

- To configure the connector to use a schema definition stored in the MongoDB database that you are connecting to, in the **Mechanism** drop-down list, select **Database**.

> **Note:**
> For information about how to create a schema definition using the Schema Editor application, see the *Schema Editor User Guide* located in the installation directory of the connector.
>
> - in Windows 7 or earlier, the guide is available from the **Simba MongoDB ODBC Driver** program group in the Start menu.
> - in Windows 8 or later, you can search for the guide on the Start screen.

15. Use the options in the Sampling area to configure how the connector samples data to generate temporary schema definitions:
    a. In the **Sampling Method** list, select the sampling method to use. Select **Forward** to sample data sequentially starting from the first record in the database, or select **Backward** to sample sequentially from the last record, or select **Random** to sample a randomly selected set of records.
    b. In the **Documents to sample** field, type the maximum number of documents that the connector can sample to generate the schema definition. To sample every document in the database, type **0**.

Important:

- Make sure to configure the connector to sample all the necessary data. Documents that are not sampled do not get included in the schema definition, and consequently do not become available in ODBC applications.
- Typically, sampling a large number of documents results in a schema definition that is more accurate and better able to represent all the data in the database. However, the sampling process may take longer than expected when many documents are sampled, especially if the database contains complex, nested data structures.
- The sampling options set here are for fallback sampling by the connector only. The sampling behavior of the Schema Editor application is set separately.

16. In the **Step Size** field, type the interval at which the connector samples a record when scanning through the database. For example, if you set this option to **2**, then the connector samples every second record in the database starting from the first record.
17. To configure write-back behavior in the connector, click **Writeback Options**. For more information, see Configuring Write-Back Options in Windows.
18. To save your settings and close the Advanced Options dialog box, click **OK**.
19. To close the Simba MongoDB ODBC Driver DSN Setup dialog box, click **OK**.

Configuring Write-Back Options in Windows

You can configure write-back options to modify how the connector writes data to the MongoDB data store.

To configure write-back options in Windows:

1. To access write-back options, open the ODBC Data Source Administrator where you created the DSN, select the DSN, click **Configure**, click **Advanced Options**, and then click **Writeback Options**.
2. In the **Batch Size** field, type the maximum number of documents that the connector can handle at one time during a write operation.
3. Use the options in the Write Concern Settings area to configure how the connector reports the success of a write operation:
   a. In the **Write Concern** field, type the total number of primary and secondary servers that must acknowledge a write operation in order for the connector to report a successful write operation.

> **Note:**
> - If you use a value that is greater than 1, make sure to also specify an appropriate value in the **Timeout** field. Setting this option to a value greater than 1 without specifying a timeout interval may cause the connector to wait indefinitely for replica set members to come online.
> - The process for acknowledging a write operation typically takes four times longer than an INSERT or UPDATE operation, but it is necessary if fault tolerance is important.

b. In the **Timeout** field, type the maximum number of seconds that the connector waits for a secondary server to acknowledge a write operation before reporting that the operation has failed.

c. To require the data to be committed to the journal before a write operation can be acknowledged, select the **Journaled Writes** check box.

> **Note:**
> For detailed information about the write concern feature in MongoDB, see "Write Concern" in the *MongoDB Manual*: https://docs.mongodb.org/manual/core/write-concern/.

4. To save your settings and close the Writeback Options dialog box, click **OK**.
5. Click **OK** to close the Advanced Options dialog box, and then click **OK** to close the Simba MongoDB ODBC Driver DSN Setup dialog box.

Configuring SSL Verification in Windows

If you are connecting to a MongoDB server that has Secure Sockets Layer (SSL) enabled, then you can configure the connector to connect to an SSL-enabled socket. When connecting to a server over SSL, the connector supports identity verification between the client and the server.

Configuring an SSL Connection without Identity Verification

You can configure a connection that uses SSL but does not verify the identity of the client or the server.

To configure an SSL connection without verification in Windows:

1. To access the SSL options for a DSN, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **SSL Options**.
2. Select the **Enable SSL** check box.
3. Select the **Allow Self-Signed Certificates** check box.
4. To save your settings and close the dialog box, click **OK**.

Configuring One-way SSL Verification

You can configure one-way verification so that the client verifies the identity of the MongoDB server.

To configure one-way SSL verification in Windows:

1. To access the SSL options for a DSN, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **SSL Options**.
2. Select the **Enable SSL** check box.
3. Choose one:
   - To verify the server using a certificate from a specific PEM file, in the **Certificate Authority File** field, specify the full path of the PEM file.
   - Or, to verify the server using certificates stored in multiple PEM files, in the **Certificate Authority Directory** field, specify the full path to the directory where the PEM files are located.
4. In the **Certificate Revocation List File** field, specify the full path of the PEM file containing the list of revoked certificates.
5. To save your settings and close the dialog box, click **OK**.

Configuring Two-way SSL Verification

You can configure two-way SSL verification so that the client and the MongoDB server verify each other.

To configure two-way SSL verification in Windows:

1. To access the SSL options for a DSN, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **SSL Options**.
2. Select the **Enable SSL** check box.
3. In the **PEM Key File** field, specify the full path of the PEM file containing the certificate for verifying the client.
4. If the client certificate is protected with a password, type the password in the **PEM Key Password** field.
5. Choose one:
   - To verify the server using a certificate from a specific PEM file, in the **Certificate Authority File** field, specify the full path of the PEM file.
   - Or, to verify the server using certificates stored in multiple PEM files, in the **Certificate Authority Directory** field, specify the full path to the directory where the PEM files are located.
6. In the **Certificate Revocation List File** field, specify the full path of the PEM file containing the list of revoked certificates.
7. To save your settings and close the dialog box, click **OK**.

Configuring FIPS in Windows

You can configure the FIPS (Federal Information Processing Standards) module to ensure the security, quality, and processing compatibility of various services.

To configure FIPS in Windows:

1. Unzip the OpenSSL_3.0_Modules_Windows_vs2015.zip Windows package released with the connector.
2. Follow the instructions mentioned in the **README.md** file.

> **Note:**
> - Make sure that all the FIPS module binary files are present in the OPENSSL_MODULES path, otherwise the connector does not work as expected.
> - When connecting to MongoDB with FIPSMode enabled, the SCRAM-SHA-1 authentication mechanism cannot be used.

Exporting a Data Source Name in Windows

After you configure a DSN, you can export it to be used on other machines. When you export a DSN, all of its configuration settings are saved in a `.sdc` file. You can then distribute the `.sdc` file to other users so that they can import your DSN configuration and use it on their machines.

To export a Data Source Name in Windows:

1. Open the ODBC Data Source Administrator where you created the DSN, select the DSN, click **Configure**, and then click **Logging Options**.
2. Click **Export Configuration**, specify a name and location for the exported DSN, and then click **Save**.

   Your DSN is saved as a `.sdc` file in the location that you specified.

Importing a Data Source Name in Windows

You can import a DSN configuration from a `.sdc` file and then use those settings to connect to your data source.

To import a Data Source Name in Windows:

1. Open the ODBC Data Source Administrator where you created the DSN, select the DSN, click **Configure**, and then click **Logging Options**.
2. Click **Import Configuration**, browse to select the `.sdc` file that you want to import the DSN configuration from, and then click **Open**.
3. Click **OK** to close the Logging Options dialog box.

   The SimbaMongoDB ODBC Driver DSN Setup dialog box loads the configuration settings from the selected `.sdc` file. You can now save this DSN and use it to connect to your data source.

Configuring Logging Options in Windows

To help troubleshoot issues, you can enable logging. In addition to functionality provided in the Simba Simba MongoDB ODBC Connector, the ODBC Data Source Administrator provides tracing functionality.

Important: Only enable logging or tracing long enough to capture an issue. Logging or tracing decreases performance and can consume a large quantity of disk space.

Configuring Connector-wide Logging Options

The settings for logging apply to every connection that uses the Simba MongoDB ODBC Connector, so make sure to disable the feature after you are done using it. To configure logging for the current connection, see Configuring Logging for the Current Connection.

To enable connector-wide logging in Windows:

1. To access logging options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Logging Options**.
2. From the **Log Level** drop-down list, select the logging level corresponding to the amount of information that you want to include in log files:

Logging Level	Description
OFF	Disables all logging.
FATAL	Logs severe error events that lead the connector to abort.
ERROR	Logs error events that might allow the connector to continue running.
WARNING	Logs events that might result in an error if action is not taken.
INFO	Logs general information that describes the progress of the connector.
DEBUG	Logs detailed information that is useful for debugging the connector.
TRACE	Logs all connector activity.

3. In the **Log Path** field, specify the full path to the folder where you want to save log files. You can type the path into the field, or click **Browse** and then browse to select the folder.
4. In the **Max Number Files** field, type the maximum number of log files to keep.

   Note: After the maximum number of log files is reached, each time an additional file is created, the connector deletes the oldest log file.

5. In the **Max File Size** field, type the maximum size of each log file in megabytes (MB).

   Note: After the maximum file size is reached, the connector creates a new file and continues logging.

6. Click **OK**.
7. Restart your ODBC application to make sure that the new settings take effect.

The Simba MongoDB ODBC Connector produces the following log files at the location you specify in the Log Path field:

- A `mongodbodbcdriver.log` file that logs connector activity that is not specific to a connection.

- A `simbamongodbodbcdriver_connection_[Number].log` file for each connection made to the database, where *[Number]* is a number that identifies each log file. This file logs connector activity that is specific to the connection.

If you enable the `UseLogPrefix` connection property, the connector prefixes the log file name with the user name associated with the connection and the process ID of the application through which the connection is made. For more information, see UseLogPrefix.

To disable connector logging in Windows:

1. Open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Logging Options**.
2. From the **Log Level** drop-down list, select **LOG_OFF**.
3. Click **OK**.
4. Restart your ODBC application to make sure that the new settings take effect.

Configuring Logging for the Current Connection

You can configure logging for the current connection by setting the logging configuration properties in the DSN or in a connection string. For information about the logging configuration properties, see Configuring Logging Options in Windows. Settings in the connection string take precedence over settings in the DSN, and settings in the DSN take precedence over connector-wide settings.

Note: If the LogLevel configuration property is passed in via the connection string or DSN, the rest of the logging configurations are read from the connection string or DSN and not from the existing connector-wide logging configuration.

To configure logging properties in the DSN, you must modify the Windows registry. For information about the Windows registry, see the Microsoft Windows documentation.

Important: Editing the Windows Registry incorrectly can potentially cause serious, system-wide problems that may require re-installing Windows to correct.

To add logging configurations to a DSN in Windows:

1. Choose one:
   - If you are using Windows 7 or earlier, click **Start**, then type **regedit** in the **Search** field, and then click **regedit.exe** in the search results.
   - Or, if you are using Windows 8 or later, on the Start screen, type **regedit**, and then click the **regedit** search result.
2. Navigate to the appropriate registry key for the bitness of your connector and your machine:
   - 32-bit System DSNs: **HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\ODBC\ODBC.INI*[DSN Name]***

- 64-bit System DSNs: **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI*[DSN Name]***
- 32-bit and 64-bit User DSNs: **HKEY_CURRENT_USER\SOFTWARE\ODBC\ODBC.INI\ *[DSN Name]***

3. For each configuration option that you want to configure for the current connection, create a value by doing the following:
   a. If the key name value does not already exist, create it. Right-click the *[DSN Name]* and then select **New > String Value**, type the key name of the configuration option, and then press **Enter**.
   b. Right-click the key name and then click **Modify**.

      To confirm the key names for each configuration option, see Connector Configuration Options.
   c. In the Edit String dialog box, in the **Value Data** field, type the value for the configuration option.
4. Close the Registry Editor.
5. Restart your ODBC application to make sure that the new settings take effect.

Verifying the Connector Version Number in Windows

If you need to verify the version of the Simba MongoDB ODBC Connector that is installed on your Windows machine, you can find the version number in the ODBC Data Source Administrator.

To verify the connector version number in Windows:

1. From the Start menu, go to **ODBC Data Sources**.

   Note: Make sure to select the ODBC Data Source Administrator that has the same bitness as the client application that you are using to connect to MongoDB.
2. Click the **Drivers** tab and then find the Simba MongoDB ODBC Connector in the list of ODBC Connectors that are installed on your system. The version number is displayed in the **Version** column.

macOS Connector

This section provides an overview of the Connector in the mac OS platform, outlining the required system specifications and the steps for installing and configuring the connector in mac OS environments.

macOS System Requirements

Install the connector on client machines where the application is installed. Each client machine that you install the connector on must meet the following minimum system requirements:

- 500MB of available disk space
- One of the following ODBC driver managers installed:
  - iODBC 3.52.9 or later
  - unixODBC 2.3.6 or later
- Before you can use the Kerberos authentication mechanism, you must install Kerberos. For information about how to install and configure Kerberos, see the MIT Kerberos Documentation: http://web.mit.edu/kerberos/krb5-latest/doc/.

Installing the Connector in macOS

If you did not obtain this connector from the Simba website, you might need to follow a different installation procedure. For more information, see the *Simba OEM ODBC Connectors Installation Guide*.

The Simba MongoDB ODBC Connector is available for macOS as a `.dmg` file named `Simba MongoDB <Version Number>.dmg`. The connector supports 64-bit client applications only.

To install the Simba MongoDB ODBC Connector in macOS:

1. Double-click **SimbaMongoDBODBC.dmg** to mount the disk image.
2. Double-click **SimbaMongoDB ODBC.pkg** to run the installer.
3. In the installer, click **Continue**.
4. On the Software License Agreement screen, click **Continue**, and when the prompt appears, click **Agree** if you agree to the terms of the License Agreement.
5. Optionally, to change the installation location, click **Change Install Location**, then select the desired location, and then click **Continue**.

   > **Note:** By default, the connector files are installed in the `/Library/simba/mongodbodbc` directory.

6. To accept the installation location and begin the installation, click **Install**.
7. When the installation completes, click **Close**.

8. If you received a license file through email, then copy the license file into the `/lib` subfolder in the connector installation directory. You must have root privileges when changing the contents of this folder.

   For example, if you installed the connector to the default location, you would copy the license file into the `/Library/simba/mongodb/lib` folder.

   Next, configure the environment variables on your machine to make sure that the ODBC Driver manager can work with the connector. For more information, see Configuring the ODBC Driver Manager in Non-Windows Machines

Verifying the Connector Version Number in macOS

If you need to verify the version of the Simba MongoDB ODBC Connector that is installed on your macOS machine, you can query the version number through the Terminal.

To verify the connector version number in macOS:

- At the Terminal, run the command:

```
pkgutil --info com.simba.mongodbodbc
```

The command returns information about the Simba MongoDB ODBC Connector that is installed on your machine, including the version number.

Linux Connector

This section provides an overview of the Connector in the Linux platform, outlining the required system specifications and the steps for installing and configuring the connector in Linux environments.

The Linux connector is available as an RPM file and as a tarball package.

Linux System Requirements

Install the connector on client machines where the application is installed. Each client machine that you install the connector on must meet the following minimum system requirements:

- 800MB of available disk space
- One of the following ODBC driver managers installed:
  - iODBC 3.52.9 or later
  - unixODBC 2.2.14 or later
- Before you can use the Kerberos authentication mechanism, you must install Kerberos. For information about how to install and configure Kerberos, see the MIT Kerberos Documentation: http://web.mit.edu/kerberos/krb5-latest/doc/.

> **Note:** The 32-bit variant of the connector is not supported on Linux ARM platforms and Red Hat® Enterprise Linux® (RHEL) 10 (ARM and Intel).

To install the connector, you must have root access on the machine.

Installing the Connector Using the RPM File

If you did not obtain this connector from the Simba website, you might need to follow a different installation procedure. For more information, see the *Simba OEM ODBC Connectors Installation Guide*.

On 64-bit editions of Linux, you can execute both 32- and 64-bit applications. However, 64-bit applications must use 64-bit connectors, and 32-bit applications must use 32-bit connectors. Make sure that you use a connector whose bitness matches the bitness of the client application:

- `simbamongodb-[Version]-[Release].i686.rpm` for the 32-bit connector
- `simbamongodb-[Version]-[Release].x86_64.rpm` for the 64-bit connector

The placeholders in the file names are defined as follows:

- *[Version]* is the version number of the connector.
- *[Release]* is the release number for this version of the connector.

To install the Simba MongoDB ODBC Connector using the RPM File:

1. Log in as the root user.
2. Navigate to the folder containing the RPM package for the connector.
3. Depending on the Linux distribution that you are using, run one of the following commands from the command line, where *[RPMFileName]* is the file name of the RPM package:
   - If you are using Red Hat Enterprise Linux or CentOS, run the following command:

     ```
     yum --nogpgcheck localinstall [RPMFileName]
     ```

   - Or, if you are using SUSE Linux Enterprise Server, run the following command:

     ```
     zypper install [RPMFileName]
     ```

4. If you received a license file through email, then copy the license file into the `/opt/simba/mongodb/lib/32` or `/opt/simba/mongodb/lib/64` folder, depending on the version of the connector that you installed.

Next, configure the environment variables on your machine to make sure that the ODBC Driver manager can work with the connector. For more information, see Configuring the ODBC Driver Manager in Non-Windows Machines

Installing the Connector Using the Tarball Package

If you did not obtain this connector from the Simba website, you might need to follow a different installation procedure. For more information, see the *Simba OEM ODBC Connectors Installation Guide*.

The Simba MongoDB ODBC Connector is available as a tarball package named `SimbaMongoDBODBC-[Version].[Release]-Linux.tar.gz`, where *[Version]* is the version number of the connector and *[Release]* is the release number for this version of the connector. The package contains both the 32-bit and 64-bit versions of the connector.

On 64-bit editions of Linux, you can execute both 32- and 64-bit applications. However, 64-bit applications must use 64-bit connectors, and 32-bit applications must use 32-bit connectors. Make sure that you use a connector whose bitness matches the bitness of the client application. You can install both versions of the connector on the same machine.

To install the connector using the tarball package:

1. Log in as the root user, and then navigate to the folder containing the tarball package.
2. Run the following command to extract the package and install the connector:

   ```
   tar  --directory=/opt -zxvf [TarballName]
   ```

   Where `[TarballName]` is the name of the tarball package containing the connector.

The Simba MongoDB ODBC Connector files are installed in the `/opt/simba/mongodbodbc` directory.

3. If you received a license file through email, then copy the license file into the `opt/simba/mongodb/lib/32` or `opt/simba/mongodb/lib/64` folder, depending on the version of the connector that you installed.

Next, configure the environment variables on your machine to make sure that the ODBC Driver manager can work with the connector. For more information, see Configuring the ODBC Driver Manager in Non-Windows Machines.

Verifying the Connector Version Number in Linux

If you need to verify the version of the Simba MongoDB ODBC Connector that is installed on your Linux machine, you can query the version number through the command-line interface if the connector was installed using an RPM file. Alternatively, you can search the connector's binary file for version number information.

To verify the connector version number in Linux using the command-line interface:

- Depending on your package manager, at the command prompt, run one of the following commands:
  - yum list | grep SimbaMongoDBODBC
  - rpm -qa | grep SimbaMongoDBODBC

The command returns information about the Simba MongoDB ODBC Connector that is installed on your machine, including the version number.

To verify the connector version number in Linux using the binary file:

1. Navigate to the `/lib` subfolder in your connector installation directory. By default, the path to this directory is: `/opt/simba/mongodbodbc/lib`.
2. Open the connector's `.so` binary file in a text editor, and search for the text `$driver_version_sb$:`. The connector's version number is listed after this text.

Configuring FIPS in Linux

You can configure the FIPS (Federal Information Processing Standards) module to ensure the security, quality, and processing compatibility of various services.

To configure FIPS in Linux:

1. Unzip the OpenSSL_3.0_Modules_Linux_gcc5_5.tar.gz Linux package released with the connector.
2. Follow the instructions mentioned in the **README.md** file.

Note:

- Make sure that all the FIPS module binary files are present in the OPENSSL_MODULES path, otherwise the connector does not work as expected.
- When connecting to the MongoDB with FIPSMode enabled, the SCRAM-SHA-1 authentication mechanism cannot be used.

AIX Connector

This section provides an overview of the Connector in the AIX platform, outlining the required system specifications and the steps for installing and configuring the connector in AIX environments.

AIX System Requirements

Install the connector on client machines where the application is installed. Each machine that you install the connector on must meet the following minimum system requirements:

- 150 MB of available disk space
- One of the following ODBC driver managers installed:
  - iODBC 3.52.9 or later
  - unixODBC 2.2.14 or later
- Before you can use the Kerberos authentication mechanism, you must install Kerberos. For information about how to install and configure Kerberos, see the MIT Kerberos Documentation: http://web.mit.edu/kerberos/krb5-latest/doc/.

To install the connector, you must have root access on the machine.

Note: The Schema Editor application is not supported on AIX, so you cannot create and save new schema definitions. When connecting to a database, the AIX connector must use a temporary schema definition that is generated during connection time or a persistent schema definition that was created on another platform.

Installing the Connector on AIX

The Simba MongoDB ODBC Connector is available for 64-bit editions of AIX as a tarball package named `SimbaMongoDBODBC-[Version].[Release]-AIX.tar.gz`, where *[Version]* is the version number of the connector, and *[Release]* is the release number for this version of the connector. The AIX connector does not support 32-bit client applications.

To install the Simba MongoDB ODBC Connector on AIX:

1. Log in as the root user, and then navigate to the folder containing the tarball package.
2. Run the following command to extract the package and install the connector:

   tar --directory=/opt -zxvf *[TarballName]*

   Where `[TarballName]` is the name of the tarball package containing the connector.

   The Simba MongoDB ODBC Connector files are installed in the `opt/simba/mongodb` directory.
3. If you received a license file through email, then copy the license file into the `opt/simba/mongodb/lib/64` folder. You must have root privileges when changing the contents of this folder.

Next, configure the environment variables on your machine to make sure that the ODBC driver manager can work with the connector. For more information, see Configuring the ODBC Driver Manager in Non-Windows Machines.

Configuring the ODBC Driver Manager in Non-Windows Machines

To make sure that the ODBC Driver manager on your machine is configured to work with the Simba MongoDB ODBC Connector, do the following:

- Set the library path environment variable to make sure that your machine uses the correct ODBC Driver manager. For more information, see Specifying ODBC Driver Managers in Non-Windows Machines .
- If the connector configuration files are not stored in the default locations expected by the ODBC driver manager, then set environment variables to make sure that the Driver manager locates and uses those files. For more information, see Specifying the Locations of the Connector Configuration Files.

  After configuring the ODBC Driver manager, you can configure a connection and access your data store through the connector.

Specifying ODBC Driver Managers in Non-Windows Machines

You need to make sure that your machine uses the correct ODBC Driver manager to load the connector. To do this, set the library path environment variable.

macOS

If you are using a macOS machine, then set the DYLD_LIBRARY_PATH environment variable to include the paths to the ODBC driver manager libraries. For example, if the libraries are installed in `/usr/local/lib`, then run the following command to set DYLD_LIBRARY_PATH for the current user session:

```
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/usr/local/lib
```

For information about setting an environment variable permanently, refer to the macOS shell documentation.

Linux

If you are using a Linux machine, then set the LD_LIBRARY_PATH environment variable to include the paths to the ODBC driver manager libraries. For example, if the libraries are installed in `/usr/local/lib`, then run the following command to set LD_LIBRARY_PATH for the current user session:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

For information about setting an environment variable permanently, refer to the Linux shell documentation.

AIX

If you are using an AIX machine, then set the LIBPATH environment variable to include the paths to the ODBC driver manager libraries. For example, if the libraries are installed in `/usr/local/lib`, then run the following command to set LIBPATH for the current user session:

```
export LIBPATH=$LD_LIBRARY_PATH:/usr/local/lib
```

For information about setting an environment variable permanently, refer to the AIX shell documentation.

Specifying the Locations of the Connector Configuration Files

By default, ODBC Driver managers are configured to use hidden versions of the `odbc.ini` and `odbcinst.ini` configuration files (named `.odbc.ini` and `.odbcinst.ini`) located in the home directory, as well as the `simba.mongodbodbc.ini` file in the `lib` subfolder of the connector installation directory. If you store these configuration files elsewhere, then you must set the environment variables described below so that the driver manager can locate the files.

If you are using iODBC, do the following:

- Set ODBCINI to the full path and file name of the `odbc.ini` file.
- Set ODBCINSTINI to the full path and file name of the `odbcinst.ini` file.
- Set SIMBA_MONGODB_ODBC_INI to the full path and file name of the `simba.mongodbodbc.ini` file.

If you are using unixODBC, do the following:

- Set ODBCINI to the full path and file name of the `odbc.ini` file.
- Set ODBCSYSINI to the full path of the directory that contains the `odbcinst.ini` file.
- Set SIMBA_MONGODB_ODBC_INI to the full path and file name of the `simba.mongodbodbc.ini` file.

For example, if your `odbc.ini` and `odbcinst.ini` files are located in `/usr/local/odbc` and your `simba.mongodbodbc.ini` file is located in `/etc`, then set the environment variables as follows:

For iODBC:

```
export ODBCINI=/usr/local/odbc/odbc.ini
export ODBCINSTINI=/usr/local/odbc/odbcinst.ini
export SIMBA_MONGODB_ODBC_INI=/etc/simba.mongodbodbc.ini
```

For unixODBC:

```
export ODBCINI=/usr/local/odbc/odbc.ini
export ODBCSYSINI=/usr/local/odbc
export SIMBA_MONGODB_ODBC_INI=/etc/simba.mongodbodbc.ini
```

To locate the `simba.mongodbodbc.ini` file, the connector uses the following search order:

1. If the SIMBA_MONGODB_ODBC_INI environment variable is defined, then the connector searches for the file specified by the environment variable.
2. The connector searches the directory that contains the connector library files for a file named `simba.mongodbodbc.ini`.
3. The connector searches the current working directory of the application for a file named `simba.mongodbodbc.ini`.
4. The connector searches the home directory for a hidden file named `simba.mongodbodbc.ini` (prefixed with a period).
5. The connector searches the `/etc` directory for a file named `simba.mongodbodbc.ini`.

Configuring ODBC Connections in Non-Windows Machine

The following sections describe how to configure ODBC connections when using the Simba MongoDB ODBC Connector on non-Windows platforms:

- Creating a Data Source Name in a Non-Windows Machine
- Configuring a DSN-less Connection on a Non-Windows Machine
- Configuring Authentication on a Non-Windows Machine
- Configuring SSL Verification on a Non-Windows Machine
- Configuring Logging Options in a Non-Windows Machine
- Testing the Connection in Non-Windows Machine

Creating a Data Source Name in a Non-Windows Machine

When connecting to your data store using a DSN, you only need to configure the `odbc.ini` file. Set the properties in the `odbc.ini` file to create a DSN that specifies the connection information for your data store. For information about configuring a DSN-less connection instead, see Configuring a DSN-less Connection on a Non-Windows Machine.

If your machine is already configured to use an existing `odbc.ini` file, then update that file by adding the settings described below. Otherwise, copy the `odbc.ini` file from the `Setup` subfolder in the connector installation directory to the home directory, and then update the file as described below.

To create a Data Source Name on a non-Windows machine:

1. In a text editor, open the `odbc.ini` configuration file.

   > **Note:** If you are using a hidden copy of the `odbc.ini` file, you can remove the period (`.`) from the start of the file name to make the file visible while you are editing it.

2. In the `[ODBC Data Sources]` section, add a new entry by typing a name for the DSN, an equal sign (=), and then the name of the connector.

   For example, on a macOS machine:

   [ODBC Data Sources]

   Sample DSN=Simba MongoDB ODBC Connector

   As another example, for a 64-bit connector on a Linux or AIX machine:

   [ODBC Data Sources]

   Sample DSN=Simba MongoDB ODBC Connector 64-bit

3. Create a section that has the same name as your DSN, and then specify configuration options as key-value pairs in the section:
   a. Set the `Driver` property to the full path of the connector library file that matches the bitness of the application.

      For example, on a macOS machine:

      Driver=/Library/simba/mongodb/lib/libmongodbodbc_sbu.dylib

      As another example, for a 64-bit connector on a Linux or AIX machine:

      Driver=/opt/simba/mongodb/lib/64/libmongodbodbc_sb64.so

   b. Set the `Server` property to the IP address or host name of the server, and then set the `Port` property to the number of the TCP port that the server uses to listen for client connections.

      For example:

      Server=192.168.222.160

      Port=27017

   c. Set the `Database` property to the name of the database that you want to access.

      For example:

      Database=TestData

   d. If authentication is required to access the server, then specify the authentication mechanism and your credentials. For more information, see Configuring Authentication on a Non-Windows Machine.

   e. If you want to connect to the server through SSL, then enable SSL and specify the certificate information. For more information, see Configuring SSL Verification on a Non-Windows Machine.

   f. If you are using a specific schema definition for this connection, do one of the following:

      - To use a schema definition that is stored in the database, set the `LoadMetadataTable` property to `1`.

      - Or, to use a schema definition that is stored in a JSON file, set the `LoadMetadataTable` property to `0` and the `LocalMetadataFile` property to the full path and name of the JSON file.

      For example, if your schema definition is stored in a JSON file:

      LoadMetadataTable=0

      LocalMetadataFile=/localhome/simba/schemas/mongodb_schema.json

Note:

- If you connect to the data store without specifying a schema definition, and the data store does not already contain a valid schema definition, then the connector automatically generates a temporary schema definition in order to support the connection.
- For information about how to create a schema definition using the Schema Editor application, see the *Schema Editor User Guide* located in the installation directory of the connector. Note that the Schema Editor is not supported on AIX.

g. Optionally, set additional key-value pairs as needed to specify other optional connection settings. For detailed information about all the configuration options supported by the Simba MongoDB ODBC Connector, see Connector Configuration Properties.

4. Save the `odbc.ini` configuration file.

Note:
If you are storing this file in its default location in the home directory, then prefix the file name with a period (`.`) so that the file becomes hidden. If you are storing this file in another location, then save it as a non-hidden file (without the prefix), and make sure that the ODBCINI environment variable specifies the location. For more information, see Specifying the Locations of the Connector Configuration Files.

For example, the following is an `odbc.ini` configuration file for macOS containing a DSN that connects to MongoDB without authentication and without a specific schema definition:

[ODBC Data Sources]

Sample DSN=Simba MongoDB ODBC Connector

[Sample DSN]

Driver=/Library/simba/mongodb/lib/libmongodbodbc_sbu.dylib

Server=192.168.222.160

Port=27017

Database=TestData

As another example, the following is an `odbc.ini` configuration file for a 64-bit connector on a Linux or AIX machine, containing a DSN that connects to MongoDB without authentication and without a specific schema definition:

[ODBC Data Sources]

Sample DSN=Simba MongoDB ODBC Connector 64-bit

[Sample DSN]

Driver=/opt/simba/mongodb/lib/64/libmongodbodbc_sb64.so

Server=192.168.222.160

Port=27017

Database=TestData

You can now use the DSN in an application to connect to the data store.

Configuring a DSN-less Connection on a Non-Windows Machine

To connect to your data store through a DSN-less connection, you need to define the connector in the `odbcinst.ini` file and then provide a DSN-less connection string in your application.

If your machine is already configured to use an existing `odbcinst.ini` file, then update that file by adding the settings described below. Otherwise, copy the `odbcinst.ini` file from the `Setup` subfolder in the connector installation directory to the home directory, and then update the file as described below.

To define a connector on a non-Windows machine:

1. In a text editor, open the `odbcinst.ini` configuration file.

   Note:
   If you are using a hidden copy of the `odbcinst.ini` file, you can remove the period (`.`) from the start of the file name to make the file visible while you are editing it.

2. In the `[ODBC Drivers]` section, add a new entry by typing a name for the connector, an equal sign (=), and then `Installed`.

   For example:

   [ODBC Drivers]

   Simba MongoDB ODBC Connector=Installed

3. Create a section that has the same name as the connector (as specified in the previous step), and then specify the following configuration options as key-value pairs in the section:
   a. Set the `Driver` property to the full path of the connector library file that matches the bitness of the application.

      For example, on a macOS machine:

      Driver=/Library/simba/mongodb/lib/libmongodbodbc_sbu.dylib

      As another example, for a 64-bit connector on a Linux or AIX machine:

      Driver=/opt/simba/mongodb/lib/64/libmongodbodbc_sb64.so
   b. Optionally, set the `Description` property to a description of the connector.

      For example:

      Description=Simba MongoDB ODBC Connector

4. Save the `odbcinst.ini` configuration file.

   > **Note:**
   > If you are storing this file in its default location in the home directory, then prefix the file name with a period (`.`) so that the file becomes hidden. If you are storing this file in another location, then save it as a non-hidden file (without the prefix), and make sure that the ODBCINSTINI or ODBCSYSINI environment variable specifies the location. For more information, see Specifying the Locations of the Connector Configuration Files.

   For example, the following is an `odbcinst.ini` configuration file for macOS:

   [ODBC Drivers]

   Simba MongoDB ODBC Connector=Installed

   [Simba MongoDB ODBC Connector]

   Description=Simba MongoDB ODBC Connector

   Driver=/Library/simba/mongodb/lib/libmongodbodbc_sbu.dylib

   As another example, the following is an `odbcinst.ini` configuration file for both the 64-bit connector in Linux or AIX:

   [ODBC Drivers]

   Simba MongoDB ODBC Connector 64-bit=Installed

   [Simba MongoDB ODBC Connector 64-bit]

   Description=Simba MongoDB ODBC Connector (64-bit)

   Driver=/opt/simba/mongodb/lib/64/libmongodbodbc_sb64.so

   You can now connect to your data store by providing your application with a connection string where the `Driver` property is set to the connector name specified in the `odbcinst.ini` file, and all the other necessary connection properties are also set. For more information, see "DSN-less Connection String Examples" in Using a Connection String.

   For instructions about configuring specific connection features, see the following:

   - Configuring Authentication on a Non-Windows Machine
   - Configuring SSL Verification on a Non-Windows Machine

For detailed information about all the connection properties that the connector supports, see Connector Configuration Properties.

Note:

- If you connect to the data store without specifying a schema definition, and the data store does not already contain a valid schema definition, then the connector automatically generates a temporary schema definition in order to support the connection. For information about specifying a schema definition, see Mechanism (Metadata) and Local File.
- For information about how to create a schema definition using the Schema Editor application, see the *Schema Editor User Guide* located in the installation directory of the connector. Note that the Schema Editor is not supported on AIX.

Configuring Authentication on a Non-Windows Machine

Some MongoDB databases require authentication. You can configure the Simba MongoDB ODBC Connector to authenticate your connection to provide your credentials and authenticate the connection to the database using one of the following methods:

- Using SCRAM-SHA-1
- Using SCRAM-SHA-256
- Using Kerberos
- Using LDAP

Note:
The MONGO-CR authentication mechanism is deprecated as of MongoDB version 3.0.

The Simba MongoDB ODBC Connector officially supports MongoDB 3.6 through 4.2 only, but still provides limited support for the MONGO-CR authentication mechanism. If authentication through SCRAM-SHA-1 fails, the connector automatically retries authentication using MONGO-CR instead, potentially enabling connections to MongoDB 2.x.

You can set the connection properties described below in a connection string or in a DSN (in the `odbc.ini` file). Settings in the connection string take precedence over settings in the DSN.

Using SCRAM-SHA-1

You can configure the connector to use the SCRAM-SHA-1 protocol to authenticate the connection. SCRAM-SHA-1 is the default authentication protocol used by MongoDB.

Note:

If authentication through SCRAM-SHA-1 fails, the connector automatically retries authentication using the MONGO-CR mechanism instead. MONGO-CR is deprecated as of MongoDB version 3.0.

To configure SCRAM-SHA-1 authentication on a non-Windows machine:

1. Set the `AuthMechanism` property to `SCRAM-SHA-1`.
2. To use a database other than the admin database to check your credentials, set the `AuthSource` property to the name of the database.
3. Set the `UID` property to an appropriate user name for accessing the MongoDB database.
4. Set the `PWD` property to the password corresponding to the user name you typed above.

Using SCRAM-SHA-256

> **Note:**
>
> SCRAM-SHA-256 authentication is only supported on MongoDB version 4.0 and above.

You can configure the connector to use the SCRAM-SHA-256 protocol to authenticate the connection.

To configure SCRAM-SHA-256 authentication on a non-Windows machine:

1. Set the `AuthMechanism` property to `SCRAM-SHA-256`.
2. To use a database other than the admin database to check your credentials, set the `AuthSource` property to the name of the database.
3. Set the `UID` property to an appropriate user name for accessing the MongoDB database.
4. Set the `PWD` property to the password corresponding to the user name you typed above.

Using Kerberos

You can configure the connector to use the Kerberos protocol to authenticate the connection.

Kerberos must be installed and configured before you can use this authentication mechanism. For information about how to install and configure Kerberos, see the MIT Kerberos Documentation: http://web.mit.edu/kerberos/krb5-latest/doc/.

To configure Kerberos authentication on a non-Windows machine:

1. Set the `AuthMechanism` property to `GSSAPI`.
2. Set the `gssapiServiceName` property to the service name of the MongoDB server.

Using LDAP

You can configure the connector to use the LDAP protocol to authenticate the connection.

To configure LDAP authentication on a non-Windows machine:

1. Set the `AuthMechanism` property to `PLAIN`.
2. Set the `UID` property to an appropriate user name for accessing the MongoDB database.
3. Set the `PWD` property to the password corresponding to the user name you typed above.

Configuring SSL Verification on a Non-Windows Machine

If you are connecting to a MongoDB server that has Secure Sockets Layer (SSL) enabled, then you can configure the connector to connect to an SSL-enabled socket. When connecting to a server over SSL, the connector supports identity verification between the client and the server.

You can set the connection properties described below in a connection string or in a DSN (in the `odbc.ini` file). Settings in the connection string take precedence over settings in the DSN.

Configuring an SSL Connection without Identity Verification

You can configure a connection that uses SSL but does not verify the identity of the client or the server.

To configure an SSL connection without verification on a non-Windows machine:

1. Set the `SSL` property to `1`.
2. Set the `sslAllowInvalidCertificates` property to `1`.

Configuring One-way SSL Verification

You can configure one-way verification so that the client verifies the identity of the MongoDB server.

To configure one-way SSL verification on a non-Windows machine:

1. Set the `SSL` property to `1`.
2. Choose one:

- To verify the server using a certificate from a specific `.pem` file, set the `sslCAFile` property to the full path of the PEM file.
- Or, to verify the server using certificates stored in multiple `.pem` files, set the `sslCADir` property to the full path of the directory where the PEM files are located.

3. Set the `sslCRLFile` to the full path of the `.pem` file containing the list of revoked certificates.

Configuring Two-way SSL Verification

You can configure two-way SSL verification so that the client and the MongoDB server verify each other.

To configure two-way SSL verification on a non-Windows machine:

1. Set the `SSL` property to `1`.
2. Set the `sslPEMKeyFile` property to the full path of the `.pem` file containing the certificate for verifying the client.
3. If the client certificate is protected with a password, set the `sslPEMKeyPwd` property to the password.
4. Choose one:

- To verify the server using a certificate from a specific `.pem` file, set the `sslCAFile` property to the full path of the PEM file.

- Or, to verify the server using certificates stored in multiple `.pem` files, set the `sslCADir` property to the full path of the directory where the `.pem` files are located.

5. Set the `sslCRLFile` to the full path of the `.pem` file containing the list of revoked certificates.

Configuring Logging Options in a Non-Windows Machine

To help troubleshoot issues, you can enable logging in the connector.

Important: Only enable logging long enough to capture an issue. Logging decreases performance and can consume a large quantity of disk space.

Logging is configured through connector-wide settings in the `simba.mongodbodbc.ini` file, which apply to all connections that use the connector.

You can set the connection properties described below in a connection string, in a DSN (in the `odbc.ini` file), or as a connector-wide setting (in the `simba.mongodbodbc.ini` file). Settings in the connection string take precedence over settings in the DSN, and settings in the DSN take precedence over connector-wide settings.

To enable logging on a non-Windows machine:

1. To specify the level of information to include in log files, set the `LogLevel` property to one of the following numbers:

LogLevel Value	Description
0	Disables all logging.
1	Logs severe error events that lead the connector to abort.
2	Logs error events that might allow the connector to continue running.
3	Logs events that might result in an error if action is not taken.
4	Logs general information that describes the progress of the connector.
5	Logs detailed information that is useful for debugging the connector.
6	Logs all connector activity.

2. Set the `LogPath` key to the full path to the folder where you want to save log files.
3. Set the `LogFileCount` key to the maximum number of log files to keep.

   Note: After the maximum number of log files is reached, each time an additional file is created, the connector deletes the oldest log file.

4. Set the `LogFileSize` key to the maximum size of each log file in bytes.

   Note: After the maximum file size is reached, the connector creates a new file and continues logging.

5. Optionally, to prefix the log file name with the user name and process ID associated with the connection, set the `UseLogPrefix` property to `1`.
6. Save the `simba.mongodbodbc.ini` configuration file.
7. Restart your ODBC application to make sure that the new settings take effect.

The Simba MongoDB ODBC Connector produces the following log files at the location you specify using the `LogPath` key:

- A `simbamongodbodbcdriver.log` file that logs connector activity that is not specific to a connection.
- A `simbamongodbodbcdriver_connection_[Number].log` file for each connection made to the database, where *[Number]* is a number that identifies each log file. This file logs connector activity that is specific to the connection.

If you set the `UseLogPrefix` property to `1`, then each file name is prefixed with `[UserName]_[ProcessID]_`, where *[UserName]* is the user name associated with the connection and *[ProcessID]* is the process ID of the application through which the connection is made. For more information, see UseLogPrefix.

To disable logging on a non-Windows machine:

1. Set the `LogLevel` key to `0`.
2. Save the `simba.mongodbodbc.ini` configuration file.
3. Restart your ODBC application to make sure that the new settings take effect.

Testing the Connection in Non-Windows Machine

To test the connection, you can use an ODBC-enabled client application. For a basic connection test, you can also use the test utilities that are packaged with your driver manager installation. For example, the iODBC driver manager includes simple utilities called `iodbctest` and `iodbctestw`. Similarly, the unixODBC driver manager includes simple utilities called isql and iusql.

Using the iODBC Driver Manager

You can use the iodbctest and iodbctestw utilities to establish a test connection with your connector. Use iodbctest to test how your connector works with an ANSI application, or use iodbctestw to test how your connector works with a Unicode application.

> **Note:** There are 32-bit and 64-bit installations of the iODBC driver manager available. If you have only one or the other installed, then the appropriate version of iodbctest (or iodbctestw) is available. However, if you have both 32- and 64-bit versions installed, then you need to make sure that you are running the version from the correct installation directory.

For more information about using the iODBC driver manager, see http://www.iodbc.org.

To test your connection using the iODBC driver manager:

1. Run **iodbctest** or **iodbctestw**.
2. Optionally, if you do not remember the DSN, then type a question mark (`?`) to see a list of available DSNs.
3. Type the connection string for connecting to your data store, and then press ENTER. For more information, see Using a Connection String.

   If the connection is successful, then the `SQL>` prompt appears.

Using the unixODBC Driver Manager

You can use the isql and iusql utilities to establish a test connection with your connector and your DSN. isql and iusql can only be used to test connections that use a DSN. Use isql to test how your connector works with an ANSI application, or use iusql to test how your connector works with a Unicode application.

Note: There are 32-bit and 64-bit installations of the unixODBC driver manager available. If you have only one or the other installed, then the appropriate version of isql (or iusql) is available. However, if you have both 32- and 64-bit versions installed, then you need to make sure that you are running the version from the correct installation directory.

For more information about using the unixODBC driver manager, see http://www.unixodbc.org.

To test your connection using the unixODBC driver manager:

- Run isql or iusql by using the corresponding syntax:

  - `isql` *`[DataSourceName]`*
  - `iusql` *`[DataSourceName]`*

`[DataSourceName]` is the DSN that you are using for the connection.

If the connection is successful, then the `SQL>` prompt appears.

Note: For information about the available options, run isql or iusql without providing a DSN.

Using a Connection String

For some applications, you might need to use a connection string to connect to your data source. For detailed information about how to use a connection string in an ODBC application, refer to the documentation for the application that you are using.

The connection strings in the following sections are examples showing the minimum set of connection attributes that you must specify to successfully connect to the data source. Depending on the configuration of the data source and the type of connection you are working with, you might need to specify additional connection attributes. For detailed information about all the attributes that you can use in the connection string, see Connector Configuration Properties.

DSN Connection String Example

The following is an example of a connection string for a connection that uses a DSN:

DSN=*[DataSourceName]*

[DataSourceName] is the DSN that you are using for the connection.

You can set additional configuration options by appending key-value pairs to the connection string. Configuration options that are passed in using a connection string take precedence over configuration options that are set in the DSN.

DSN-less Connection String Examples

Some applications provide support for connecting to a data source using a connector without a DSN. To connect to a data source without using a DSN, use a connection string instead.

Important:
When you connect to the data store using a DSN-less connection string, the connector does not encrypt your credentials.

The placeholders in the examples are defined as follows, in alphabetical order:

- *[MongoDBDatabase]* is the database that you want to access.
- *[PortNumber]* is the number of the TCP port that the MongoDB server uses to listen for client connections.
- *[ServerInfo]* is the IP address or host name of the MongoDB server to which you are connecting.
- *[ServiceName]* is the Kerberos service principal name of the MongoDB server.
- *[SetName]* is the name of the replica set that you want to access.
- *[YourPassword]* is the password corresponding to your user name.
- *[YourUserName]* is the user name that you use to access the MongoDB server.

Connecting to a Standard MongoDB Server Without Authentication

The following is the format of a DSN-less connection string for a standard connection to a MongoDB server that does not require authentication:

Driver=Simba MongoDB ODBC Driver;Server=*[ServerInfo]*;
Port=*[PortNumber]*;Database=*[MongoDBDatabase]*;

For example:

Driver=Simba MongoDB ODBC Driver;Server=192.168.222.160;
Port=27017;Database=TestData;

Connecting to a Standard MongoDB Server that Requires SCRAM-SHA-1 Authentication

The following is the format of a DSN-less connection string for a standard connection to a MongoDB server that requires SCRAM-SHA-1 authentication:

Driver=Simba MongoDB ODBC Driver;Server=*[ServerInfo]*;
Port=*[PortNumber]*;Database=*[MongoDBDatabase]*;
AuthMechanism=SCRAM-SHA-1;UID=*[YourUserName]*;
PWD=*[YourPassword]*;

For example:

Driver=Simba MongoDB ODBC Driver;Server=192.168.222.160;
Port=27017;Database=TestData;AuthMechanism=SCRAM-SHA-1;
UID=simba;PWD=simba;

Connecting to a Standard MongoDB Server that Requires SCRAM-SHA-256 Authentication

The following is the format of a DSN-less connection string for a standard connection to a MongoDB server that requires SCRAM-SHA-256 authentication:

Driver=Simba MongoDB ODBC Driver;Server=*[ServerInfo]*;
Port=*[PortNumber]*;Database=*[MongoDBDatabase]*;
AuthMechanism=SCRAM-SHA-256;UID=*[YourUserName]*;
PWD=*[YourPassword]*;

For example:

Driver=Simba MongoDB ODBC Driver;Server=192.168.222.160;
Port=27017;Database=TestData;AuthMechanism=SCRAM-SHA-256;
UID=simba;PWD=simba;

Connecting to a Standard MongoDB Server that Requires Kerberos Authentication

The following is the format of a DSN-less connection string for a standard connection to a MongoDB server that requires Kerberos authentication:

Driver=Simba MongoDB ODBC Driver;Server=*[ServerInfo]*;
Port=*[PortNumber]*;Database=*[MongoDBDatabase]*;

AuthMechanism=GSSAPI;gssapiServiceName=*[ServiceName]*;

For example:

Driver=Simba MongoDB ODBC Driver;Server=192.168.222.160;
Port=27017;Database=TestData;AuthMechanism=GSSAPI;
gssapiServiceName=mongodb;

Connecting to a Standard MongoDB Server that Requires LDAP Authentication

The following is the format of a DSN-less connection string for a standard connection to a MongoDB server that requires LDAP authentication:

Driver=Simba MongoDB ODBC Driver;Server=*[ServerInfo]*;
Port=*[PortNumber]*;Database=*[MongoDBDatabase]*;
AuthMechanism=PLAIN;UID=*[YourUserName]*;PWD=*[YourPassword]*;

For example:

Driver=Simba MongoDB ODBC Driver;Server=192.168.222.160;
Port=27017;Database=TestData;AuthMechanism=PLAIN;
UID=simba;PWD=simba;

Connecting to a Replica Set Without Authentication

The following is the format of a DSN-less connection string for connecting to a replica set that does not require authentication:

Driver=Simba MongoDB ODBC Driver;Server=*[ServerInfo]*;
Port=*[PortNumber]*;Database=*[MongoDBDatabase]*;
EnableReplicaSet=1;SecondaryServers=*[Server1]*:*[PortNumber1]*, *[Server2]*:
[PortNumber2];ReplicaSet=*[SetName]*;

For example:

Driver=Simba MongoDB ODBC Driver;Server=192.168.222.160;
Port=27017;Database=TestData;EnableReplicaSet=1;
SecondaryServers=192.168.222.165:27017, 192.168.222.231:27017;ReplicaSet=TestSet;

Connecting to a Replica Set that Requires MongoDB User Name and Password Authentication

The following is the format of a DSN-less connection string for connecting to a replica set that requires MongoDB User Name and Password authentication:

Driver=Simba MongoDB ODBC Driver;Server=*[ServerInfo]*;
Port=*[PortNumber]*;Database=*[MongoDBDatabase]*;
UID=*[YourUserName]*;PWD=*[YourPassword]*;EnableReplicaSet=1;
SecondaryServers=*[Server1]*:*[PortNumber1]*, *[Server2]*:*[PortNumber2]*;ReplicaSet=*[SetName]*;

For example:

Driver=Simba MongoDB ODBC Driver;Server=192.168.222.160;
Port=27017;Database=TestData;UID=simba;PWD=simba;
EnableReplicaSet=1;SecondaryServers=192.168.222.165:27017,

192.168.222.231:27017;ReplicaSet=TestSet;

Connecting to a Replica Set that Requires Kerberos Authentication

The following is the format of a DSN-less connection string for connecting to a replica set that requires Kerberos authentication:

Driver=Simba MongoDB ODBC Driver;Server=*[ServerInfo]*;
Port=*[PortNumber]*;Database=*[MongoDBDatabase]*;
AuthMechanism=GSSAPI;gssapiServiceName=*[ServiceName]*;
EnableReplicaSet=1;SecondaryServers=*[Server1]*:*[PortNumber1]*, *[Server2]*:
[PortNumber2];ReplicaSet=*[SetName]*;

For example:

Driver=Simba MongoDB ODBC Driver;Server=192.168.222.160;
Port=27017;Database=TestData;AuthMechanism=GSSAPI;
gssapiServiceName=*ServiceName*;EnableReplicaSet=1;
SecondaryServers=192.168.222.165:27017, 192.168.222.231:27017;ReplicaSet=TestSet;

Connecting to a Replica Set that Requires LDAP Authentication

The following is the format of a DSN-less connection string for connecting to a replica set that requires LDAP authentication:

Driver=Simba MongoDB ODBC Driver;Server=*[ServerInfo]*;
Port=*[PortNumber]*;Database=*[MongoDBDatabase]*;
AuthMechanism=PLAIN;UID=*[YourUserName]*;PWD=*[YourPassword]*;
EnableReplicaSet=1;SecondaryServers=*[Server1]*:*[PortNumber1]*, *[Server2]*:
[PortNumber2];ReplicaSet=*[SetName]*;

For example:

Driver=Simba MongoDB ODBC Driver;Server=192.168.222.160;
Port=27017;Database=TestData;AuthMechanism=PLAIN;
UID=simba;PWD=simba;EnableReplicaSet=1;
SecondaryServers=192.168.222.165:27017, 192.168.222.231:27017;ReplicaSet=TestSet;

Features

For more information on the features of the Simba MongoDB ODBC Connector, see the following:

- Catalog Support
- Double-Buffering
- SQL Connector
- Data Types
- Schema Definitions
- Virtual Tables
- Write-back
- Security and Authentication

Catalog Support

The Simba MongoDB ODBC Connector supports catalogs by using the name of the MongoDB database as the catalog. Doing so allows the connector to work easily with various ODBC applications.

The connector supports queries against MongoDB databases outside of the one that you are connected to, as well as joins between tables that belong to different databases. To work with data from other databases, in your SQL statements, specify the database (or catalog) that each table belongs to using the syntax *DatabaseName*.*TableName*.

Double-Buffering

The Simba MongoDB ODBC Connector is capable of using double-buffering to improve connector performance during SELECT operations.

The impact of double-buffering depends on how the transfer speed of your network compares to the data processing speed of the connector. If the transfer speed is significantly higher, then enabling double-buffering allows the connector to make full use of the network's capabilities. Conversely, if the transfer speed is considerably lower, the additional processes involved in double-buffering might cause a decrease in performance.

To make optimal use of double-buffering, you need to set an appropriate buffer size. A buffer size that is too small might decrease performance, while a buffer size that is too large might diminish the performance improvements from double-buffering. If the transfer speed of the network is slow, the additional time spend processing a large buffer size might cause a decrease in performance.

For information about configuring double-buffering, see Enable Double-Buffering.

SQL Connector

The SQL Connector feature of the connector allows applications to execute standard SQL queries against MongoDB. It converts SQL-92 queries to MongoDB API calls and processes the results.

Data Types

The Simba MongoDB ODBC Connector supports many MongoDB data types, and converts data between MongoDB and SQL data types as needed. The mapping between each data type is determined by the schema definition, which you can create by using the Schema Editor application that is installed with the connector.

Note:
The Schema Editor is not supported on AIX. On all other platforms, information about how to use the Schema Editor can be found in the *Schema Editor User Guide* located in the installation directory of the connector.

- in Windows 7 or earlier, the guide is available from the **Simba MongoDB ODBC Driver** program group in the Start menu.
- in Windows 8 or later, you can search for the guide on the Start screen.

The following table lists the supported data type mappings.

To support complex data types such as objects and arrays, the connector renormalizes the data into virtual tables. For more information, see Virtual Tables.

MongoDB Type	SQL Type
Binary	SQL_BIGINT SQL_BIT SQL_DOUBLE SQL_INTEGER SQL_VARCHAR
Boolean	
Date	SQL_TIMESTAMP
DBPointer	SQL_VARCHAR
Decimal	SQL_DECIMAL
JavaScript	SQL_VARCHAR
JavaScript (with scope)	SQL_VARCHAR **Important:** MongoDB cannot represent JavaScript (with scope) data as a string. The connector converts the data to type SQL_VARCHAR, but returns the following string as the value: `Unsupported JavaScript with Scope`.

MongoDB Type	SQL Type
MaxKey	SQL_VARCHAR
MinKey	SQL_VARCHAR
NumberDouble	SQL_BIGINT SQL_DOUBLE SQL_INTEGER SQL_VARCHAR
NumberInt	
NumberLong	
ObjectID	SQL_VARCHAR
Regular Expression	SQL_VARCHAR
String	SQL_VARCHAR
Symbol	SQL_VARCHAR
Timestamp	SQL_VARCHAR
Undefined	SQL_VARCHAR
UUID	SQL_GUID
Array	N/A The data is renormalized into a virtual table.
Object	N/A The data is renormalized into a virtual table.

Different rows in the sampled data might have the same field assigned to different data types. The connector resolves this mixed data typing by specifying a single type for the field in the schema definition. The specified type is the first data type from this list that appears in the sampled data.

1. **Array**

   Fields that have Object as one of the data types are also returned as Array data.

2. **Binary**

3. **String**

   Fields that have any of the following types as one of the data types, but not Array or Binary, are also returned as String data:

   - Date
   - Timestamp
   - DBPointer
   - JavaScript
   - JavaScript (with scope)
   - Symbol
   - Regular Expression

- MaxKey
- MinKey
- Undefined
- OID

4. **NumberDouble**
5. **NumberLong**
6. **NumberInt**

For example, the connector treats the following field named f as MongoDB data of type Array:

{f : {g1 : 1}}

{f : [1, 2, 3]}

Data types that do not have a direct mapping from MongoDB to ODBC are represented as type VARCHAR in ODBC. The detected MongoDB type is used during INSERT and UPDATE operations.

Note:

The default precision for SQL_TYPE_TIMESTAMP is `3`.

The MongoDB-type ISODate that is mapped to SQL_TYPE_TIMESTAMP stores the timestamp value in the number of milliseconds since the Unix epoch (January 1, 1970). As the value is in milliseconds, the precision cannot be greater than 3.

To revert to the previous SQL_TYPE_TIMESTAMP behavior (precision is set to `6`, and precision in schema maps is ignored), append the connection string as follows:

```
IgnoreTimestampPrecisionFromSchemaMap=1;DefaultTimestampPrecision=6;
```

Be aware that if you revert to the previous behavior, the connector does not warn about the timestamp truncation.

Embedded Documents

The connector renormalizes embedded documents into columns. For example, consider the following JSON document:

{"contact": {"address": {"street": "1-123 Broadway", "city": "Vancouver"}}, "phone": "+12345678"}

When generating the schema definition, the connector identifies the following columns in the document, all of type String:

contact_address_street	contact_address_city	phone
1-123 Broadway	Vancouver	+12345678

The connector is able to work with these columns as if they were standard table columns.

Arrays

The ODBC interface does not natively support collection-based data types, so the Simba MongoDB ODBC Connector implements two options for accessing and interacting with collection-based data. Depending on preference, arrays in MongoDB can be renormalized into virtual tables or columns.

By default, arrays are renormalized into virtual tables. To view array elements as columns instead, use the **Move to Parent** option in the Schema Editor. You can create the column in a parent virtual table or in the base table.

Note:
The Schema Editor is not supported on AIX. On all other platforms, information about how to use the Schema Editor can be found in the *Schema Editor User Guide* located in the installation directory of the connector.

- in Windows 7 or earlier, the guide is available from the **SimbaMongoDB ODBC Driver** program group in the Start menu.
- in Windows 8 or later, you can search for the guide on the Start screen.

Arrays as Virtual Tables

The connector can renormalize MongoDB arrays into virtual tables. For more information, see Virtual Tables.

Arrays as Columns

The connector can also renormalize MongoDB arrays into columns. Consider the following JSON document:

{"values": ["hello", 1, {"v1": {"v2": "this is an embedded document"}}]}

The connector can represent the array elements using the following columns, where values_1 is of type Double and the other two are of type String:

values_0	values_1	values_2_v1_v2
Hello	1.0	this is an embedded document

The connector works with these columns as if they were standard table columns.

The column names include the index of the array element that the column represents, starting with an index of 0. In other words, the first element of the array uses a suffix of _0, the second element uses _1, and so on.

Schema Definitions

To enable consistent support for your MongoDB data, you must configure the connector to use a schema definition from a JSON file or the database. You can use the Schema Editor application to create a schema definition and then save it in a JSON file or the database.

Note:

- in Windows 7 or earlier, the guide is available from the **SimbaMongoDB ODBC Driver** program group in the Start menu.
- in Windows 8 or later, you can search for the guide on the Start screen.

The Schema Editor is not supported on AIX. On all other platforms, information about how to use the Schema Editor can be found in the *Schema Editor User Guide* located in the installation directory of the connector.

When the connector connects to a database without a specified schema definition, it automatically generates a temporary schema definition using the settings defined in the Sampling area of the Advanced Options dialog box or the `SamplingStrategy`, `SamplingLimit`, and `SamplingStepSize` properties. However, temporary schema definitions do not persist after the connection is closed, and the connector may generate different schema definitions during subsequent connections to the same database due to variances in the data that gets sampled.

Important: When creating a schema definition or connecting to the database without specifying one, make sure to configure the connector to sample all the necessary data. Documents that are not sampled do not get included in the schema definition, and consequently do not become available in ODBC applications. If the schema definition was created through the Schema Editor, you can use the Schema Editor to sample additional documents and add them to the schema definition. Note that the Schema Editor is not supported on AIX.

Mapping MongoDB Data

MongoDB is able to store data that follows different rules of data typing and structure compared to traditional relational data, and the tables may contain complex data such as nested arrays or arrays of differently-typed elements. Because traditional ODBC toolsets may not support these data structures, the Simba MongoDB ODBC Connector generates a schema definition that maps the MongoDB data to an ODBC-compatible format.

The Simba MongoDB ODBC Connector does the following when generating a schema definition:

1. Samples the data in the database in order to detect its structure and determine the data mappings that best support the data.
2. Assigns a MongoDB data type to each column.
3. Maps each MongoDB data type to the SQL data type that is best able to represent the greatest number of values.
4. Renormalizes single-level objects into columns.
5. For each array and nested object in the database, the connector generates a virtual table to expand the data, and saves these virtual tables as part of the schema definition. For more information about virtual tables, see Virtual Tables.

During this sampling process, the connector defines data types for each column, but does not change the data types of the individual cells in the database. As a result, columns may contain mixed data types. During read operations, values are converted to match the SQL data type of the column so that the connector can work with all the data in the column consistently.

Virtual Tables

One advantage of the MongoDB design is the ability to store data that is denormalized into a fewer number of tables. However, the ODBC interface does not natively support accessing denormalized data types such as arrays and objects. By expanding the data contained within arrays and objects into virtual tables, the Simba MongoDB ODBC Connector allows you to directly interact with the data but leave the storage of the data in its denormalized form in MongoDB.

If any columns are mapped to a denormalized data type during the sampling process, then the connector creates the following tables and saves them as part of the schema definition:

- A base table, which contains only the normal data from the real table.
- A virtual table for each column of denormalized data, expanding the nested data.

Virtual tables refer to the data in the real table, enabling the connector to access the denormalized data. By querying the virtual tables, you can access the contents of arrays and objects via ODBC. When you write or modify data in a virtual table, the data in the real table in the MongoDB database is updated.

The base table and virtual tables appear as additional tables in the list of tables that exist in the database, and are named using the following conventions:

- The base table uses the same name as the real table that it represents.
- In the ODBC layer, the name of each virtual table is formed using the name of the real table that contains the array or object, an underscore (_), and the name of the array or object.
- In the MongoDB layer, the name of each virtual table is formed using the name of the collection that the data comes from, a period (`.`), and then the name of the array or object followed by a set of closed square brackets (`[]`) for each hierarchy level in which the array or object is nested.
- If a virtual table or column has the same name as an actual table or column in the database, then the connector appends _1 to the virtual table or column name as a suffix. The number increments as necessary until the name is unique.

For example, consider the example MongoDB table named CustomerTable shown below.

_id	Customer Name	Invoices	Service Level	Contacts	Ratings
1111	ABC	[{invoice_id: ”123”, item: ”toaster”, price: ”456”, discount: ”0.2”}, {invoice_id: ”124”, item: ”oven”, price: ”1235”, discount: ”0.2”}]	Silver	[{type: primary, name: "John Johnson"}, {type: invoicing, name: "Jill Jilliamson"}]	[5,6]
2222	XYZ	[{invoice_id: ”135”, item: ”fridge”, price: ”12543”, discount: ”0.0”}]	Gold	[{type: primary, name: "Jane Doe"}]	[1,2]

CustomerTable has two columns that have an array of Objects in each cell, Invoices and Contacts, and one column that has an array of Scalar types, Ratings. Multiple virtual tables would be generated for this single source table. The first table is the base table, which is shown below.

_id	Customer Name	Service Level
1111	ABC	Silver
2222	XYZ	Gold

The base table contains all of the data of the original table, but the data from the arrays has been omitted and will be expanded in the virtual tables.

The following three tables show the virtual tables that represent the original arrays in the example.

_id	Invoices_dim1_idx	invoice_id	item	price	discount
1111	0	123	toaster	456	0.2
1111	1	124	oven	1235	0.2
2222	0	135	fridge	12543	0.0

_id	Contacts_dim1_idx	type	name
1111	0	primary	John Johnson
1111	1	invoicing	Jill Jilliamson
2222	0	primary	Jane Doe

_id	Ratings_dim1_idx	Ratings_value
1111	0	5
1111	1	6
2222	0	1
2222	1	2

Each of these tables contain the following:

- A reference back to the original primary key column corresponding to the row of the original array (via the _id column)
- An indication of the position of the data within the original array (using the Invoices_dim1_idx, Contacts_dim1_idx, and Ratings_dim1_idx columns)
- The expanded data for each element within the array:
  - **Invoices**: invoice_id, item, price and discount
  - **Contacts**: type and name
  - **Ratings**: Ratings_value

You can select, insert, and update data in the base tables and virtual tables as if they were standard relational tables, and the connector will handle the storage details within MongoDB.

For example, to append 7 to the Ratings array in the CustomerTable where _id = 1111, execute the following statement:

INSERT INTO CustomerTable_Ratings (_id, Ratings_value) VALUES (1111, 7)

The following examples show supported cases for passdown of the LIKE filter and are based on the following document in MongoDB:

_id	one_dim_array	nested_arrays	multi_dim_array
Row1	["pears", "oranges"]	[{ "fruit": "pear", "varieties": ["Bosc", "Bartlett"] }, { "fruit": "orange", "varieties": ["Navel", "Cara Cara"] }]	[[{ "fruit": "pear", "varieties": ["Bosc", "Bartlett"] }], [{ "fruit": "orange", "varieties": ["Navel", "Cara Cara"] }]]

The LIKE filter is on an _id column:

SELECT * FROM fruits_nested_arrays_varieties WHERE _id LIKE 'r%'

The LIKE filter is on a column that is nested within a one-dimensional array:

SELECT * FROM fruits_one_dim_array WHERE fruits_one_dim_array LIKE 'p%'

SELECT * FROM fruits_nested_arrays_varieties WHERE fruits_nested_arrays_varieties LIKE '%a%'

The LIKE filter is on an _id column or a column that is nested within a one-dimensional array and is combined with an AND condition:

SELECT * FROM fruits_nested_arrays_varieties WHERE fruits_nested_arrays_varieties LIKE '%a%' AND _id = 'Row1'

SELECT * FROM fruits_nested_arrays_varieties WHERE _id LIKE '%Row1' AND fruits_nested_arrays_varieties != 'Bosc'

The LIKE filter is on an _id column or a column that is nested within a one-dimensional array and is combined with an OR condition:

SELECT * FROM fruits_nested_arrays_varieties WHERE fruits_nested_arrays_varieties LIKE '%a%' OR fruits_nested_arrays_varieties LIKE 'b%'

SELECT * FROM fruits_nested_arrays_varieties WHERE fruits_nested_arrays_varieties LIKE '%a%' or _id = 'Row1'

A LIKE query on a column that is nested within a multi-dimentional array cannot be passed down to MongoDB:

SELECT * FROM fruits_multi_dim_array_dim2_varieties WHERE fruits_multi_dim_array_dim2_varieties LIKE '%a%'

The above query will not be passed down to MongoDB, but the connector will handle the query to return the expected result.

A query containing an OR condition between LIKE and non-LIKE filters on virtual table cannot be passed down to MongoDB:

The MongoDB does not support passdown of non-LIKE filters on virtual tables to MongoDB. The query will not be passed down to MongoDB, but the connector will handle the query to return the expected result.

Some operations may be processed differently or not supported for certain types of data. For more information, see Write-back.

Write-back

The Simba MongoDB ODBC Connector supports Data Manipulation Language (DML) statements such as INSERT, UPDATE, and DELETE. The connector does not support SQL subqueries or ODBC transactions.

Important:
Writing data to the MongoDB database may change the typing of the data. If the existing data is of a type that is different from the column data type specified in the schema definition, then the existing data will be replaced by new data that is of the column data type.

When the connector samples the data and generates the schema definition, the connector defines a SQL data type and a MongoDB data type for each column in the base tables and virtual tables, but does not change the data types of the individual values in the database. As a result, columns may contain mixed data types. The connector supports DML statements on mixed data types. When you execute a write operation, the connector will attempt to complete the operation using the column data type specified in the schema definition.

The data provided in DML statements must match the column data types. For example, a String value cannot be inserted in a column that is defined as an Integer column in the schema definition.

The connector handles each DML statement as described below.

Important:
The CREATE, ALTER, and DROP statements are not supported for tables.

INSERT

Note:
When inserting a value into an array, do not specify the bottom-level index value in your INSERT statement. The bottom-level index value is no longer required as of connector version 2.0.0.

Each row in MongoDB needs to have a unique ID represented by the _id column. If not provided during insertion, MongoDB auto-generates a unique ID for each row. The _id field is exposed as a valid column in the connector and can be auto-generated when issuing INSERT statements through the connector. For example, consider the following table.

_id	sample_column
"517024D6CC79814E3FEBD352"	1
"5170ED77E49CC93A918DE316"	2

To insert a document with an auto-generated value for _id (data type: jstOID), issue the following command:

INSERT INTO sample_table_1(sample_column) VALUES(3)

The following table shows the table after the insertion.

_id	sample_column
"517024D6CC79814E3FEBD352"	1
"5170ED77E49CC93A918DE316"	2
"51710FFCE49CC93A918DE322"	3

The value for the _id column can also be inserted using the INSERT statements, as in the following examples:

INSERT INTO sample_table_2(_id, sample_column) VALUES(1,1)

INSERT INTO sample_table_2 VALUES(1,1)

UPDATE

When updating rows, special care needs to be taken to avoid duplicate values for the _id column. As mentioned before, _id needs to be unique across all rows. When an UPDATE statement tries to set a value for the _id column and matches multiple rows, only one of the rows is updated with the new values, and the connector return an error for the remaining rows. UPDATE is not executed atomically.

DELETE

The DELETE statement is not supported for virtual tables.

The connector considers a table valid as long as the table contains some data. If a table is completely empty, then the connector is not able to access the table. Consider the following example:

DELETE FROM sample_table_3

The command removes all data from sample_table_3. Therefore, sample_table_3 is invalid. Any users attempting to access the table receive an error.

Security and Authentication

To protect data from unauthorized access, some MongoDB data stores require connections to be authenticated with user credentials or encrypted using the SSL protocol. The Simba MongoDB ODBC Connector provides full support for these authentication protocols.

Note: In this documentation, "SSL" refers to both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The connector supports up to TLS 1.2. The SSL version used for the connection is the highest version that is supported by both the connector and the server.

The connector provides a mechanism that enables you to authenticate your connection using the SCRAM-SHA-1 protocol (which MongoDB uses by default), the Kerberos protocol, or the LDAP protocol. The connector also provides limited support for the MONGO-CR protocol, which is

deprecated as of MongoDB 3.0. For detailed configuration instructions, see Configuring Authentication in Windows or Configuring Authentication on a Non-Windows Machine.

Additionally, the connector supports the following types of SSL connections:

- No identity verification
- One-way authentication
- Two-way authentication

It is recommended that you enable SSL whenever you connect to a server that is configured to support it. SSL encryption protects data and credentials when they are transferred over the network, and provides stronger security than authentication alone. For detailed configuration instructions, see Configuring SSL Verification in Windows or Configuring SSL Verification on a Non-Windows Machine.

Pushdown Optimization

You can optimize connector performance by restricting the scope of a query to include the required columns only.

The connector supports both filter and aggregation pushdown:

- Filter pushdown applies the filter to the server, rather than retrieving all rows from the server and then applying the filter.
- Aggregation pushdown applies the aggregation to the server, rather than retrieving all rows from the server and then applying the aggregation.

Important:
Pushdown optimization is not supported when the projection of a subquery contains complex elements such as scalar functions.

For example, to perform a filter pushdown, use a query like the following:

```
SELECT "T1_1"."Column1" FROM (SELECT "OneHundredRecords"."Column1" FROM
"OneHundredRecords") AS "T1_1" WHERE "Column1" = '10'
```

As another example, to perform an aggregation pushdown, use a query like the following:

```
SELECT sum ("Column1") FROM ( SELECT "OneHundredRecords"."Column1" FROM
"OneHundredRecords" )
```

Connector Configuration Properties

Connector Configuration Options lists the configuration options available in the Simba MongoDB ODBC Connector alphabetically by field or button label. Options having only key names, that is, not appearing in the user interface of the connector, are listed alphabetically by key name.

When creating or configuring a connection from a Windows machine, the fields and buttons described below are available in the following dialog boxes:

- Simba MongoDB ODBC Driver DSN Setup
- Advanced Options
- SSL Options
- Writeback Options
- Logging Options

When using a connection string or configuring a connection from a non-Windows machine, use the key names provided below.

Configuration Options Appearing in the User Interface

The following configuration options are accessible via the Windows user interface for the Simba MongoDB ODBC Connector, or via the key name when using a connection string or configuring a connection from a Linux or macOS computer:

- Allow Self-Signed Certificates
- Authentication Source
- Batch Size
- Binary Column Size
- Certificate Authority Directory
- Certificate Authority File
- Certificate Revocation List File
- Connect to Replica Set
- Database
- Documents to Fetch Per Block
- JSON Column Size
- Local File
- Log Level
- Log Path
- Max File Size
- Max Number Files
- Mechanism (Authentication)
- Mechanism (Metadata)
- Omit Explicit NULL Columns On Insert
- Password

Allow Self-Signed Certificates

This option specifies whether the connector allows self-signed SSL certificates from the server.

- Enabled (`1`): The connector authenticates the MongoDB server even if the server is using a self-signed certificate.
- Disabled (`0`): The connector does not allow self-signed certificates from the server.

Note:
This setting is only applicable when SSL is enabled.

Key Name	Default Value	Required
`sslAllowInvalidCertificates`	Clear (`0`)	No

Authentication Source

The name of the MongoDB database that you want to use to check your credentials for authentication.

Key Name	Default Value	Required
AuthSource	admin	No

Batch Size

The maximum number of documents that the connector can handle at one time during a write operation.

Key Name	Default Value	Required
DmlBatchSize	500	No

Binary Column Size

The maximum data length for Binary columns.

Note:
The maximum value that you can set for this option is 2147483647.

Key Name	Default Value	Required
DefaultBinaryColumnLength	32767	No

Certificate Authority Directory

The full path of the directory containing the `.pem` files that you use to verify the server. This setting enables the connector to access multiple `.pem` files for SSL verification.

Note:
To specify a single `.pem` file, use the Certificate Authority File option or the `sslCAFile` key instead.

Key Name	Default Value	Required
sslCADir	None	No

Certificate Authority File

The full path of the `.pem` file that you use to verify the server.

Note:
To configure the connector to access multiple `.pem` files for SSL verification, use the Certificate Authority Directory option or the `sslCADir` key instead.

Key Name	Default Value	Required
sslCAFile	None	No

Certificate Revocation List File

The full path of the `.pem` file containing the list of revoked certificates.

Key Name	Default Value	Required
`sslCRLFile`	None	No

Connect to Replica Set

This option specifies whether the connector can access replica sets in your MongoDB implementation.

- Enabled (`1`): The connector can access replica sets in your MongoDB implementation.
- Disabled (`0`): The connector cannot access replica sets.

Key Name	Default Value	Required
`EnableReplicaSet`	Clear (`0`)	No

Database

Note: To inspect your databases and determine the appropriate schema to use, at the MongoDB command prompt, type `show databases`.

The name of the MongoDB database that you want to access.

Key Name	Default Value	Required
`Database`	`test`	Yes

Documents to Fetch Per Block

The maximum number of documents that a query returns at a time.

This setting also determines the buffer size used when double-buffering is enabled.

Key Name	Default Value	Required
`BatchSize`	`4096`	No

Documents to Sample

The maximum number of records that the connector can sample to generate a temporary schema definition. When this option is set to `0`, the connector samples every document in the database.

Important:

- Make sure to configure the connector to sample all the necessary data. Documents that are not sampled do not get included in the schema definition, and consequently do not become available in ODBC applications.
- Typically, sampling a large number of documents results in a schema definition that is more accurate and better able to represent all the data in the database. However, the sampling process may take longer than expected when many documents are sampled, especially if the database contains complex, nested data structures.

Key Name	Default Value	Required
`SamplingLimit`	`100`	No

Enable BypassDocumentValidation

This option specifies whether the connector bypasses schema validation.

- Enabled (`1`): The connector bypasses schema validation.
- Disabled (`0`): The connector does not bypass schema validation.

Key Name	Default Value	Required
`BypassDocumentValidation`	Clear (`0`)	No

Enable Double-Buffering

This option specifies whether the connector retrieves the data using double-buffering.

- Enabled (`1`): The connector retrieves the data using double-buffering.
- Disabled (`0`): The connector retrieves the data using single-buffering.

Key Name	Default Value	Required
`EnableDoubleBuffer`	Selected (`1`)	No

Enable JSON Read/Write Mode

This option specifies whether the connector reports a special column named DocumentAsJson that retrieves or stores whole documents as JSON-formatted strings.

- Enabled (`1`): The connector reports the JSON column.
- Disabled (`0`): The connector does not report the JSON column.

Note:
Only enable this option as needed. The process of converting documents into Json strings decreases the performance of the connector.

See also the connector configuration option JSON Column Size.

Key Name	Default Value	Required
UseJsonColumn	0	No

Enable Mixed Type Filter

This option specifies whether the connector uses mixed type filtering, which allows a value to pass the filter even if it is not the same data type as the specified filter term.

- Enabled (1): When filtering data, the connector retrieves all values that match the specified filter term, even if the values are stored as different data types than the filter term. For example, if the data store contains "1000" as a NumberInt value and also as a String value, when you execute a query that filters for "1000" as a String value, the connector returns both the NumberInt and String values.

- Disabled (0): When filtering data, the connector only retrieves values that match the data type of the specified filter term. For example, if the data store contains "1000" as a NumberInt value and also as a String value, when you execute a query that filters for "1000" as a String value, the connector only returns the String value.

  Note:
  Mixed type filtering requires the connector to scan the entire MongoDB collection. You can disable this feature to increase the connector's performance, but note that doing so also alters the results of your queries.

Key Name	Default Value	Required
EnableMixedTypeFilter	Selected (1)	No

Enable Passdown

This option specifies whether the connector optimizes joins between virtual tables, and passes filtering and aggregation optimizations to the MongoDB database for handling.

- Enabled (1): The connector optimizes joins between virtual tables, and passes filtering and aggregation optimizations to the MongoDB database for handling.

- Disabled (0): The connector does not optimize joins, and does not pass filtering and aggregation optimizations to the MongoDB database.

Key Name	Default Value	Required
EnablePassdownOptimization	Selected (1)	No

EnableIdxIndicatorForEmptyVT

This property controls the behavior for fetching and inserting empty arrays in virtual tables.

- 0 : Empty arrays in virtual tables are ignored during fetching, and inserting empty arrays is not enabled. This is the default option.
- 1 : Empty arrays in virtual tables are included in the result set during fetching, with rows using a bottom-level index of -3 to indicate an empty array. Inserting empty arrays is enabled by setting -3 as the bottom-level index.

Key Name	Default Value	Required
EnableIdxIndicatorForEmptyVT	0	No

EnableIdxIndicatorForNullVT

This property controls the behavior for fetching and inserting null arrays in virtual tables.

- 0 : Null arrays in virtual tables are ignored during fetching, and inserting null arrays is not enabled. This is the default option.
- 1 : Null arrays in virtual tables are included in the result set during fetching, with rows using a bottom-level index of -4 to indicate a null array. Inserting null arrays is enabled by setting -4 as the bottom-level index.

Key Name	Default Value	Required
EnableIdxIndicatorForNullVT	0	No

EnableRelaxedExtendedJson

The EnableRelaxedExtendedJson property allows you to specify the JSON format for the DocumentAsJson column.

- 0 : The DocumentAsJson column uses the canonical JSON format. This is the default option.
- 1 : The DocumentAsJson column uses the relaxed extended JSON format.

Note:

- The EnableRelaxedExtendedJson property is only applicable when UseJsonColumn=1.
- A $date value with a year before 1970 or after 3000 will always be displayed in the canonical format, even when EnableRelaxedExtendedJson=1, due to limitations in third-party libraries that handle date/time values.

Key Name	Default Value	Required
EnableRelaxedExtendedJson	0	No

Enable SSL

This option specifies whether the connector uses SSL to connect to the server.

- Enabled (1): The connector uses SSL to connect to the server.
- Disabled (0): The connector does not use SSL to connect to the server.

Key Name	Default Value	Required
SSL	Clear (0)	No

Enable Temporary Table Compression

This option specifies whether the Simba SQL Engine compresses the temporary generated from executing nested select queries.

- Enabled (1): The SQL Engine generates compressed temporary tables for nested select queries.
- Disabled (0): The SQL Engine generates uncompressed temporary tables for nested select queries.

Note:
If this option is enabled, the performance of nested select queries might be affected.

Key Name	Default Value	Required
EnableTempTableCompression	Disabled (0)	No

Encrypt Password

This option specifies how the connector encrypts the credentials that are saved in the DSN:

- **Current User Only**: The credentials are encrypted, and can only be used by the current Windows user.
- **All Users Of This Machine**: The credentials are encrypted, but can be used by any user on the current Windows machine.

Important:
This option is available only when you configure a DSN using the MongoDB ODBC Driver DSN Setup dialog box in the Windows connector. When you connect to the data store using a connection string, the connector does not encrypt your credentials.

Key Name	Default Value	Required
N/A	All Users Of This Machine	No

Expose Binary as SQL_LONGVARBINARY

This option specifies whether the connector returns Binary columns as data of type SQL_LONGVARBINARY or SQL_VARBINARY.

- Enabled (`1`): The connector returns Binary columns as SQL_LONGVARBINARY data.
- Disabled (`0`): The connector returns Binary columns as SQL_VARBINARY data.

Key Name	Default Value	Required
`UseSqlLongVarbinary`	`1`	No

Expose Strings as SQL_WVARCHAR

This option specifies how the String data type is mapped to SQL.

- Enabled (`1`): The MongoDBString data type is mapped to SQL_WVARCHAR.
- Disabled (`0`): The MongoDBString data type is mapped to SQL_VARCHAR.

Key Name	Default Value	Required
`UseSqlWVarchar`	Selected (`1`)	No

JSON Column Size

The default column length for JSON fields.

See also the connector configuration option Enable JSON Read/Write Mode.

Key Name	Default Value	Required
JsonColumnSize	1023	No

Journaled Writes

This option specifies whether the connector requires that the data from a write operation to be committed to the journal before the write operation can be acknowledged.

- Enabled (`1`): The connector requires that the data from a write operation to be committed to the journal before the write operation can be acknowledged.
- Disabled (`0`): The connector does not require that the data from a write operation to be committed to the journal before the write operation can be acknowledged.

Note:
This option is only applicable when the Write Concern option or the `WriteConcern` key is enabled.

Key Name	Default Value	Required
`WriteConcernJournaled`	Clear(`0`)	No

Local File

The full path of a local JSON file containing the schema definition that you want the connector to use when connecting to MongoDB.

Key Name	Default Value	Required
`LocalMetadataFile`	None	Yes, if the metadata mechanism is set to Local File (`LoadMetadataTable=0`).

Log Level

Use this property to enable or disable logging in the connector and to specify the amount of detail included in log files.

Important:

- Only enable logging long enough to capture an issue. Logging decreases performance and can consume a large quantity of disk space.
- The settings for logging apply to every connection that uses the Simba MongoDB ODBC Connector, so make sure to disable the feature after you are done using it.
- This option is not supported in connection strings. To configure logging for the Windows connector, you must use the Logging Options dialog box. To configure logging for a non-Windows connector, you must use the `simba.mongodbodbc.ini` file.

Set the property to one of the following values:

- OFF (`0`): Disable all logging.
- FATAL (`1`): Logs severe error events that lead the connector to abort.
- ERROR (`2`): Logs error events that might allow the connector to continue running.
- WARNING (`3`): Logs events that might result in an error if action is not taken.
- INFO (`4`): Logs general information that describes the progress of the connector.

- DEBUG (5): Logs detailed information that is useful for debugging the connector.
- TRACE (6): Logs all connector activity.

When logging is enabled, the connector produces the following log files at the location you specify in the Log Path (LogPath) property:

- A simbamongodbodbcdriver.log file that logs connector activity that is not specific to a connection.
- A simbamongodbodbcdriver_connection_*[Number]*.log file for each connection made to the database, where *[Number]* is a number that identifies each log file. This file logs connector activity that is specific to the connection.

If you enable the UseLogPrefix connection property, the connector prefixes the log file name with the user name associated with the connection and the process ID of the application through which the connection is made. For more information, see UseLogPrefix.

Key Name	Default Value	Required
LogLevel	OFF (0)	No

Log Path

The full path to the folder where the connector saves log files when logging is enabled.

Important: When logging with connection strings and DSNs, this option only applies to per-connection logs.

Key Name	Default Value	Required
LogPath	None	Yes, if logging is enabled.

Max File Size

The maximum size of each log file in bytes. After the maximum file size is reached, the connector creates a new file and continues logging.

If this property is set using the Windows UI, the entered value is converted from megabytes (MB) to bytes before being set.

Important: When logging with connection strings and DSNs, this option only applies to per-connection logs.

Key Name	Default Value	Required
LogFileSize	20971520	No

Max Number Files

The maximum number of log files to keep. After the maximum number of log files is reached, each time an additional file is created, the connector deletes the oldest log file.

Important: When logging with connection strings and DSNs, this option only applies to per-connection logs.

Key Name	Default Value	Required
`LogFileCount`	`50`	No

Mechanism (Authentication)

Important:
This section describes the Mechanism option from the SimbaMongoDB ODBC Driver DSN Setup dialog box. For information about the Mechanism option from the Advanced Options dialog box, see Mechanism (Metadata).

The authentication mechanism to use, selected from the following:

- No Authentication (`None`): The connector does not authenticate the connection.
- MongoDB User Name and Password (`SCRAM-SHA-1`): The connector authenticates the connection using the SCRAM-SHA-1 protocol, which is the default authentication protocol used by MongoDB.

  Note:
  If authentication through SCRAM-SHA-1 fails, the connector automatically retries authentication using the MONGO-CR mechanism instead. MONGO-CR is deprecated as of MongoDB version 3.0.

- SCRAM-SHA-256 (`SCRAM-SHA-256`): The connector authenticates the connection using the SCRAM-SHA-256 protocol.

  Note:
  SCRAM-SHA-256 authentication is only supported on MongoDB version 4.0 and above.

- Kerberos (`GSSAPI`): The connector authenticates the connection using the Kerberos protocol.
- LDAP (`PLAIN`): The connector authenticates the connection using the LDAP protocol.

Key Name	Default Value	Required
`AuthMechanism`	No Authentication (`None`)	No

Mechanism (Metadata)

Important:
This section describes the Mechanism option from the Advanced Options dialog box. For information about the Mechanism option from the SimbaMongoDB ODBC Driver DSN Setup dialog box, see Mechanism (Authentication).

This option specifies where the connector looks for the schema definition.

- Database (`1`): The connector loads the schema definition from the MongoDB database.
- Local File (`0`): The connector loads the schema definition from the JSON file specified in the Local File field or the `LocalMetadataFile` key.

Key Name	Default Value	Required
`LoadMetadataTable`	Database (`1`)	No

Omit Explicit NULL Columns On Insert

This option specifies whether the connector writes explicitly provided null values to columns during INSERT operations.

- Enabled (`1`): The connector does not write null values to the columns of a table, even if the null values are explicitly provided.
- Disabled (`0`): The connector writes explicitly provided null values to columns as specified, but no default null value is inserted.

Note:
This option only affects INSERT operations.

Key Name	Default Value	Required
`OmitColumns`	Selected (`1`)	No

Omit explicit NULL columns on update

This option specifies whether the connector writes explicitly provided null values to columns during update operations.

- Enabled (`1`): The connector executes an update operation where one or more columns have null values provided in the set clause, the null values are not written back to the server.
- Disabled (`0`): The connector executes an update operation where one or more columns have null values provided in the set clause, the null values are written back to the server.

Note:

- If `OmitNullColumnsOnUpdate` is set to `1` and either an `_id` or `virtual table index column` is set to `null` in an update operation, the connector neither throws an error nor updates as expected.
- When `UseJsonColumn` and `OmitNullColumnsOnUpdate` are set to `1` and an update operation has `DocumentAsJson` in the set clause with a `JSON` string that has one or more fields set to `null`, the null values are not ignored.

Key Name	Default Value	Required
`OmitNullColumnsOnUpdate`	Selected (`0`)	No

Password

The password corresponding to the user name that you provided in the Username field (the `UID` key).

Important: This option should not be explicitly set in the DSN properties of the Windows Registry. in Windows, if both `PWD` and `ENCRYPTED_PWD` are set, the connector always uses the value for `PWD`.

Key Name	Default Value	Required
`PWD`	None	Yes, if the authentication mechanism is MongoDB User Name and Password (`SCRAM-SHA-1`) or `SCRAM-SHA-256` or LDAP (`PLAIN`).

PEM Key File

The full path of the `.pem` file containing the certificate for verifying the client.

Key Name	Default Value	Required
`sslPEMKeyFile`	None	No

PEM Key Password

The password of the client certificate file that is specified in the PEM Key File field (`sslPEMKeyFile`).

Key Name	Default Value	Required
`sslPEMKeyPwd`	None	No

Port

The number of the TCP port that the MongoDB server uses to listen for client connections.

Key Name	Default Value	Required
`Port`	`27017`	Yes

Read Preference

Specify how the connector routes read operations to the members of a replica set. The following values are possible:

- Primary (`primary`)
- Primary preferred (`primaryPreferred`)
- Secondary (`secondary`)
- Secondary preferred (`secondaryPreferred`)
- Nearest (`nearest`)

> **Note:**
> - The values are case-sensitive.
> - This option is only available when the Connect to Replica Set option or the `EnableReplicaSet` key is enabled.

For more information, see "Replication" in the *MongoDB Manual*: http://docs.mongodb.org/manual/replication/.

Key Name	Default Value	Required
`ReadPreference`	Primary (`primary`)	No

Replica Set Name

The name of the replica set for the connector to access.

> **Note:**
> This option is only available when the Connect to Replica Set option or the `EnableReplicaSet` key is enabled.

Key Name	Default Value	Required
`ReplicaSet`	None	Yes, if connecting to a replica set.

Sampling Method

This options specifies how the connector samples data when generating a temporary schema definition.

- `Forward`: The connector samples data starting from the first record in the database, then samples the next record, and so on.
- `Backward`: The connector samples data starting from the last record in the database, then samples the preceding record, and so on.
- `Random`: The connector selects sample records from the data source at random until the sampling limit is reached (for more information, see Documents to Sample).

Note:
The random sampling strategy is only supported by MongoDB Server 3.2 or higher.

Key Name	Default Value	Required
`SamplingStrategy`	`Forward`	No

Secondary Servers

A comma-separated list of the servers that you need to use when connecting to a replica set. You can indicate the TCP port that the server is using to listen for client connections by appending a colon (`:`) and the port number to the server name or IP address.

Key Name	Default Value	Required
`SecondaryServers`	None	No

Server

The host name or IP address of the MongoDB server.

Key Name	Default Value	Required
`Server`	None	Yes

Service Name

The Kerberos service principal name of the MongoDB server.

Key Name	Default Value	Required
`gssapiServiceName`	`mongodb`	Yes, if the authentication mechanism is Kerberos.

Step Size

The interval at which the connector samples records when scanning through the database to generate a temporary schema definition. For example, if you set this option to `2`, then the connector samples every second record in the database.

If the Sampling Method is set to Random, this setting is ignored.

Key Name	Default Value	Required
`SamplingStepSize`	`1`	No

String Column Size

The maximum number of characters that can be contained in STRING columns.

Note:
The maximum value that you can set for this option is 2147483647.

Key Name	Default Value	Required
`DefaultStringColumnLength`	`255`	No

Timeout

The maximum number of seconds that the connector waits for a secondary server to acknowledge a write operation before reporting that the operation has failed.

When this option is set to `0`, the connector does not time out. Instead, the connector waits until all secondary servers acknowledge the write operation, and then reports that the operation has succeeded.

Note:
This option is only applicable when the Write Concern option or the `WriteConcern` key is enabled.

Important:
If you set the Write Concern option or key to a value that is greater than 1, make sure that this option is not set to 0. This may cause the connector to wait indefinitely for replica set members to come online.

Key Name	Default Value	Required
`WriteConcernTimeout`	`0`	No

UseLogPrefix

This option specifies whether the connector includes a prefix in the names of log files so that the files can be distinguished by user and application.

Set the property to one of the following values:

- `1`: The connector prefixes log file names with the user name and process ID associated with the connection that is being logged.

  For example, if you are connecting as a user named "jdoe" and using the connector in an application with process ID 7836, the generated log file would be named `jdoe_7836_simbamongodbodbc_driver.log`.

- `0`: The connector does not include the prefix in log file names.

  To configure this option for the Windows connector, you create a value for it in one of the following registry keys:

- For a 32-bit connector installed on a 64-bit machine: **HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Simba\Simba MongoDB ODBC Driver\Driver**

- Otherwise: **HKEY_LOCAL_MACHINE\SOFTWARE\Simba\Simba MongoDB ODBC Driver\Driver**

Use `UseLogPrefix` as the value name, and either `0` or `1` as the value data.

To configure this option for a non-Windows connector, you must use the `simba.mongodbodbc.ini` file.

Key Name	Default Value	Required
`UseLogPrefix`	`0`	No

Username

The user name that you use to access the MongoDB instance.

Key Name	Default Value	Required
`UID`	None	Yes, if the authentication mechanism is MongoDB User Name and Password (`SCRAM-SHA-1`) or `SCRAM-SHA-256` or LDAP (`PLAIN`).

Write Concern

The total number of primary and secondary servers that must acknowledge a write operation in order for the connector to report a successful write operation.

When this option is set to `0`, the connector does not require write operations to be acknowledged.

Important:

- If you set this option to a value that is greater than 1, make sure to also specify an appropriate value for the Timeout setting or the `WriteConcernTimeout` key. Setting this option to a value greater than 1 without specifying a timeout interval may cause the connector to wait indefinitely for replica set members to come online.
- The process for acknowledging a write operation typically takes four times longer than an INSERT or UPDATE operation, but it is necessary if fault tolerance is important.

Key Name	Default Value	Required
`WriteConcern`	`1`	No

Configuration Options Having Only Key Names

The following configuration options do not appear in the Windows user interface for the Simba MongoDB ODBC Connector. They are accessible only when you use a connection string or configure a connection in macOS or Linux.

- AlternativeSourceTypeAsString
- Driver
- IgnoreTransactions
- NoCursorTimeout
- ReplaceDocumentAsJsonOnUpdate
- SampleCollections
- ServerSelectionTimeoutMS
- ServerSelectionTryOnce
- SocketTimeoutMS

The `UseLogPrefix` property must be configured as a Windows Registry key value, or as a connector-wide property in the `simba.mongodbodbc.ini` file for macOS or Linux.

- UseLogPrefix

AlternativeSourceTypeAsString

This option specifies whether the connector allows String values to be inserted into the _id column when the column is of type ObjectId.

- 1: The connector allows both String and ObjectId values to be inserted into the column.
- 0: The connector only allows ObjectId values to be inserted into the column.

Key Name	Default Value	Required
AlternativeSourceTypeAsString	1	No

Driver

In Windows, the name of the installed connector for (Simba MongoDB ODBC Connector).

On other platforms, the name of the installed connector as specified in odbcinst.ini, or the absolute path of the connector shared object file.

Key Name	Default Value	Required
Driver	Simba MongoDB ODBC Connector when installed in Windows, or the absolute path of the connector shared object file when installed on a non-Windows machine.	Yes

IgnoreTransactions

This option determines whether the connector ignores attempts to perform transactions, and supports connections where auto-commit mode is disabled in the client application.

- 1: The connector ignores attempts to perform transactions, and supports connections where auto-commit mode is disabled in the client application. However, the connector does not execute the begin or commit functions even if they are explicitly called.
- 0: The connector does not ignore attempts to perform transactions. If auto-commit mode is disabled in the client application and a transaction is attempted, the connector returns an error.

Important:
Regardless of whether auto-commit mode is enabled, the connector does not wait for commit operations to complete, and always executes queries immediately.

Key Name	Default Value	Required
IgnoreTransactions	0	No

NoCursorTimeout

This option specifies whether the connector allows active cursors on the data source server to expire.

- `False`: The data source server will time out idle cursors after the threshold inactivity period set on the server.
- `True`: The connector prevents the data source server from timing out idle cursors.

Important:
When set to `True`, there is a risk that if the connector should quit or lose the connection to the server unexpectedly, the cursor will remain open on the server indefinitely. You can adjust the threshold for idle cursor timeouts on the MongoDB server, see https://docs.mongodb.com/v3.0/reference/parameters/ for details.

Key Name	Default Value	Required
`NoCursorTimeout`	`False`	No

PersistMetadata

This option specifies whether the connector persists schema maps to a file or publishes them to the MongoDB database without using the Schema Editor.

- `0`: The connector samples if the schema map does not exist, but does not persist.
- `1`: The connector samples and persists if the schema map does not exist, but does not overwrite metadata in the database or file.
- `2`: The connector samples and persists if the schema map does not exist, but overwrites metadata in the database or file.

Important:
- When set to `2`, the metadata that gets persisted and overwritten depends on the configuration of the Mechanism (Metadata) (`LoadMetadataTable`) property. For more information, see Mechanism (Metadata).

Key Name	Default Value	Required
`PersistMetadata`	`0`	No

ReplaceDocumentAsJsonOnUpdate

This option specifies whether the connector updates specified values or replaces the entire record when updating a JSON record.

- `True`: Updating a JSON record replaces the entire record. Any previously existing key-value pairs not specified in the new record are deleted.
- `False`: Updating a JSON record only affects the specified key-value pairs. Previously existing key-value pairs not specified in the new record are preserved.

Key Name	Default Value	Required
ReplaceDocumentAsJsonOnUpdate	True	No

SampleCollections

This option specifies whether the connector samples collections when generating an in-memory schema map with the Enable JSON Read/Write Mode(UseJsonColumn) property enabled.

- 1: The connector samples collections.
- 0: The connector does not sample collections. If the Enable JSON Read/Write Mode (UseJsonColumn) property is enabled, a table is generated for each encountered collection, containg two columns: _id (VARCHAR) and DocumentAsJson (VARCHAR). Virtual tables are not created in this case.

Key Name	Default Value	Required
SampleCollections	1	No

ServerSelectionTimeoutMS

The amount of time, in milliseconds, that the connector searches for a server. When this time is reached, the connector returns an error.

Note:
This property is only checked if ServerSelectionTryOnce is set to 0. If ServerSelectionTryOnce is set to 1 or is not explicitly set, the server selection timeout value is specified in the SQL_ATTR_LOGIN_TIMEOUT attribute. The default value for SQL_ATTR_LOGIN_TIMEOUT is 5000 milliseconds.

Key Name	Default Value	Required
ServerSelectionTimeoutMS	5000	No

ServerSelectionTryOnce

This option specifies whether, if the initial server selection fails, the connector searches for a server only once, or continues searching until the timeout value is reached.

- 1: If the initial server selection fails, the connector searches for a server only once, and then either selects a server or returns an error.
- 0: The connector continues searching for a server until it either selects a server or times out with an error. The default timeout value is 5000 milliseconds, and can be changed with the ServerSelectionTimeoutMS property (see ServerSelectionTimeoutMS).

Note:
If this property is set to `1` or is not explicitly set, the server selection timeout value is specified in the SQL_ATTR_LOGIN_TIMEOUT attribute. The default value for SQL_ATTR_LOGIN_TIMEOUT is 5000 milliseconds.

Key Name	Default Value	Required
`ServerSelectionTryOnce`	`1`	No

SocketTimeoutMS

The amount of time, in milliseconds, before an attempt to send or receive on a socket times out.

When this property is set to `-1`, the connector does not specify the amount of time before a socket connection times out. In this case, the timeout behavior is determined by the settings defined in the `mongo-c-driver` library, which the Simba MongoDB ODBC Connector uses as a third-party dependency.

Key Name	Default Value	Required
`SocketTimeoutMS`	`-1`	No

UseLogPrefix

This option specifies whether the connector includes a prefix in the names of log files so that the files can be distinguished by user and application.

Set the property to one of the following values:

- `1`: The connector prefixes log file names with the user name and process ID associated with the connection that is being logged.

  For example, if you are connecting as a user named "jdoe" and using the connector in an application with process ID 7836, the generated log file would be named `jdoe_7836_simbamongodbodbc_driver.log`.

- `0`: The connector does not include the prefix in log file names.

  To configure this option for the Windows connector, you create a value for it in one of the following registry keys:

- For a 32-bit connector installed on a 64-bit machine: **HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Simba\Simba MongoDB ODBC Driver\Driver**

- Otherwise: **HKEY_LOCAL_MACHINE\SOFTWARE\Simba\Simba MongoDB ODBC Driver\Driver**

Use `UseLogPrefix` as the value name, and either `0` or `1` as the value data.

To configure this option for a non-Windows connector, you must use the `simba.mongodbodbc.ini` file.

Key Name	Default Value	Required
UseLogPrefix	0	No

Upgrading from Connector Version 1.8.x

Beginning with Simba MongoDB ODBC Connector 2.2.8, the connector resumes support for schemas that were generated by the 1.8.x connectors. Connector versions 2.2.8 and later can load these schemas from SDD files or from MongoDB databases, and display MongoDB tables through ODBC the same way that the 1.8.x connectors did.

However, be aware that there are some functionality differences between the 1.8.x and 2.x connectors.

Workflow Changes Between Versions 1.8.x and 2.x

When migrating your connections from connector version 1.8.x to 2.x, be aware of the following differences:

- Using an SDD Schema in Version 2.2.8 and Later
- Inserting NULL to Automatically Generate _id Values
- Inserting an Array Value
- Using the Any Match Virtual Table

Using an SDD Schema in Version 2.2.8 and Later

Connector versions 1.8.4 and earlier save schemas in SDD format. Connector versions 2.0.0 and later save schemas in JSON format. Beginning with version 2.2.8, the connector can load and use SDD-formatted schemas in addition to JSON-formatted schemas.

However, be aware of the following:

- Compared to previous connector versions, the newer connectors report different data types for some column types. Specifically, when reporting the TYPE_NAME values for SQL data types, connector versions 1.8.4 and earlier report the BSON type names while connector versions 2.0.0 and later report the names in SQL-92 format. Also, the following column types are translated differently:

MongoDB Column Type	SQL Type Reported in Versions 1.8.4 and Earlier	SQL Type Reported in Versions 2.0.0 and Later
Array index	SQL_BIGINT	SQL_INTEGER
Binary	SQL_BINARY	SQL_VARBINARY

- Connector versions 2.0.0 and later use a different data sampling mechanism compared to connector versions 1.8.4 and earlier. It is recommended that you do not re-sample an SDD-formatted schema using a newer connector, as this can result in unusual results such as redundant tables. Instead, create a new JSON-formatted schema by sampling the database directly.

- If you make any changes to an SDD-formatted schema, it must be saved in JSON format. The connector translates the schema from SDD to JSON while preserving the table and column names defined in the schema.
- When the connector translates a schema from SDD to JSON, an "invalid source name" error may occur. For detailed information about this issue and potential workarounds for it, see below.

Invalid Source Name Error

This error is known to occur when all of the following conditions are met:

- You are using an SDD-formatted schema with connector version 2.2.8 or later.
- The MongoDB table associated with the schema has arrays that share the same name.
- These arrays contain data at different levels. For example, one is a two-level array while the other is only a single-level array.

Connector versions 2.2.8 and later handle this inconsistency in array levels by pushing the values from the upper array levels down into the bottom array level. For example, given a two-level array and a single-level array that have the same name, the connector pushes the values from the single-level array down one level.

After this push occurs, the columns that the 1.8.x connector originally generated for containing the single-level array values cause an "invalid source name" error to occur when you view the base table.

As a workaround, do one of the following:

- Re-sample the entire table using connector version 2.2.8 or later.
- Or, using the Schema Editor, hide the invalid columns.
- Or, open the schema map in a text editor and then change the source names of the invalid columns to the source names of the columns where the pushed values now reside. You cannot make this change in the Schema Editor, because the application treats this as a naming conflict.

Example

For example, a MongoDB database contains a table named Test2, which contains the following documents:

Document 1:

```
{
        "_id" : "1",
                "Col2_String" : "Hello",
                "Col3_Array" : [
                        [
```

```
                    {
                        "Int" : 01,
                        "String" : "Col3_Array_1"
                    },
                    {
                        "Int" : 02,
                        "String" : "Col3_Array_2"
                    }
                ],
            ],
}
```

Document 2:

```
{
    "_id" : "2",
    "Col2_String" : "Hello Again",
    "Col3_Array" : [
        {
            "Int" : 01,
            "String" : "Col3_Array_1"
        }
    ],
}
```

Notice that both documents contain an array named "Col3_Array". However, Col3_Array is a two-level array in Document 1, whereas it is a single-level array in Document 2.

Column #	SQL Name	SourceName
1	_id	_id
2	Col2_String	Col2_String
3	Col3_Array	Col3_Array
4	Col3_Array__0	Col3_Array.*0
5	Col3_Array__0__0	Co3_Array.*0.*0
6	Col3_Array__0__0__Int	Col3_Array.*0.*0.Int
7	Col3_Array__0__0__String	Col3_Array.*0.*0.String

Column #	SQL Name	SourceName
8	Col3_Array__0__1	Col3_Array.*0.*1
9	Col3_Array__0__1__Int	Col3_Array.*0.*1_Int
10	Col3_Array__0__1__String	Col3_Array.*0.*1_String
11	Col3_Array__0__Int	Col3_Array.*0.Int
12	Col3_Array__0__String	Col3_Array.*0.String

After the connector translates the schema definition of Test2 from SDD to JSON, the resulting base table (still named Test2) contains columns that use the following SQL Names and Source Names:

Columns 5 through 10 (the Col3_Array__0__0, Col3_Array__0__0__Int, Col3_Array__0__0__String, Col3_Array__0__1, Col3_Array__0__1__Int, and Col3_Array__0__1__String columns) were generated to contain the values from the two-level Col3_Array in Document 1.

Columns 11 and 12 (the Col3_Array__0__Int and Col3_Array__0__String columns) were originally generated to contain the values from the single-level Col3_Array in Document 2. Because of how the connector pushed the array values from columns 11 and 12 down one level, these columns now cause an "invalid source name" error when you view the base table.

To resolve this error, do one of the following:

- Re-sample the table using connector version 2.2.8 or later.
- Hide columns 11 and 12 in the Schema Editor.
- Or, use a text editor to change the source name of column 11 to Col3_Array.*0.*0.Int and the source name of column 12 to Col3_Array.*0.*0.String.

Inserting NULL to Automatically Generate _id Values

When inserting a row into a table that has an _id column of type ObjectId, you can have the connector automatically generate a valid _id value for your new row. In connector versions 1.8.4 and earlier, you do this by specifying NULL as the _id value in your query statement.

In connector versions 2.0.0 and later, you do this by writing a query that does not specify an _id value. If you specify NULL, then the connector inserts NULL into the _id column.

Example

For example, a MongoDB database contains the following table named Boolean_Table:

_id	Column1	KeyColumn
5a627dab3fc2394c4f824e94	0	false

Then, the following query is executed against Boolean_Table:

INSERT INTO Boolean_Table values (NULL, 1, 'true')

Connector versions 1.8.4 and earlier would insert a row into the table as shown below:

_id	Column1	KeyColumn
5a627dab3fc2394c4f824e94	0	false
5a6a420c012857f7e7f69f78	1	true

Connector versions 2.0.0 and later would insert a row with a NULL _id value instead, as shown below:

_id	Column1	KeyColumn
5a627dab3fc2394c4f824e94	0	false
	1	true

To automatically generate an _id value using connector versions 2.0.0 and later, insert the row by executing the following query instead:

INSERT INTO Boolean_Table(Column1, KeyColumn) values (1, 'true')

Inserting an Array Value

When inserting an array value using connector versions 1.8.4 and earlier, you must specify the bottom-level index of the array in the INSERT statement. The connector then inserts the new array value at the index position that you specified.

In connector versions 2.0.0 to 2.2.7, specifying the bottom-level index in an INSERT statement for an array value causes the connector to return an error. These versions of the connector always insert new array values in the next available position, and do not accept INSERT statements that specify the bottom-level index value.

Beginning with connector versions 2.2.8, the connector is able to successfully execute INSERT statements that contain the bottom-level index value. The connector also supports statements that specify -1 as the index. In both cases, the connector appends the new value to the end of the array, as expected. However, it is still recommended that you do not specify the bottom-level index value, since it is no longer required.

Example

For example, a MongoDB database contains a table named Test, which has a single document with the following content:

```
{
        _id : 2000,
                "Array" : [ 0,
                        {
                                "BottomLevelArray" : [
                                        {
                                                "col" : "Federer"
                                        }
```

```
                    ]
                }
            ]
}
```

The connector expands this array into the following virtual table named Test_Array_BottomLevelArray:

_id	Test_Array_dim1_idx	Test_Array_BottomLevelArray_dim1_idx	col
2000	1	0	Federer

In this example, you want to append the value "Nadal" to the end of the array (after "Federer"), so that the resulting virtual table looks like this:

_id	Test_Array_dim1_idx	Test_Array_BottomLevelArray_dim1_idx	col
2000	1	0	Federer
2000	1	1	Nadal

In connector versions 1.8.4 and earlier, you must execute the following statement:

INSERT INTO Test_Array_BottomLevelArray_dim1_idx values (2000, 1, 1, 'Nadal')

In connector versions 2.0.0 to 2.2.7, you must execute the following statement:

INSERT INTO Test_Array_BottomLevelArray_dim1_idx values (2000, 1, 'Nadal')

In connector version 2.2.8 and later, you can execute any of the following statements:

INSERT INTO Test_Array_BottomLevelArray_dim1_idx values (2000, 1, 1, 'Nadal')

INSERT INTO Test_Array_BottomLevelArray_dim1_idx values (2000, 1, 'Nadal')

INSERT INTO Test_Array_BottomLevelArray_dim1_idx values (2000, 1, -1, 'Nadal')

Note:

- Even though statements that contain the bottom-level index value are supported in connector versions 2.2.8 or later, it is still recommended that you omit the value, as it is not required.
- In all cases, you must specify the upper-level index value. This indicates which array the connector should insert the value in.

Using the Any Match Virtual Table

Connector versions 1.8.4 and earlier include support for a special virtual table called the Any Match virtual table. This table is used to support queries that filter data based on multiple top-level array values, enabling such queries to be executed using significantly shorter statements.

The Any Match virtual table is deprecated as of connector version 2.0.0, but it is still possible to re-enable and use this feature. However, note that in order to do so, you must disable the Enable Mixed Type Filter option (the `EnableMixedTypeFilter` property). For more information, see Enable Mixed Type Filter.

Third-Party Trademarks

Debian is a trademark or registered trademark of Software in the Public Interest, Inc. or its subsidiaries in Canada, United States and/or other countries.

IBM and AIX are trademarks or registered trademarks of IBM Corporation or its subsidiaries in Canada, United States, and/or other countries.

Kerberos is a trademark of the Massachusetts Institute of Technology (MIT).

Linux is the registered trademark of Linus Torvalds in Canada, United States and/or other countries.

Mac, macOS, Mac OS, and OS X are trademarks or registered trademarks of Apple, Inc. or its subsidiaries in Canada, United States and/or other countries.

Microsoft, MSDN, Windows, Windows Server, Windows Vista, and the Windows start button are trademarks or registered trademarks of Microsoft Corporation or its subsidiaries in Canada, United States and/or other countries.

Red Hat, Red Hat Enterprise Linux, and CentOS are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in Canada, United States and/or other countries.

SUSE is a trademark or registered trademark of SUSE LLC or its subsidiaries in Canada, United States and/or other countries.

Ubuntu is a trademark or registered trademark of Canonical Ltd. or its subsidiaries in Canada, United States and/or other countries.

MongoDB and Mongo are trademarks or registered trademarks of MongoDB, Inc. or its subsidiaries in Canada, the United States and/or other countries.

All other trademarks are trademarks of their respective owners.