insightsoftware

# Simba Teradata ODBC Data Connector

Installation and Configuration Guide

Version 20.00.00.028

July 2025

# Copyright

This document was released in July 2025.



## Contact Us

www.insightsoftware.com

# About This Guide

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

This guide supports the following releases:

- Teradata Database 20.00
- Simba Teradata ODBC Data Connector 20.00.00.028

This version of the Simba Teradata ODBC Connector supports the Teradata Database versions listed in the system requirements. For more information, see:

- Windows System Requirements
- macOS System Requirements
- Unix System Requirements

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

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

- Familiarity with the platform on which you are using the Simba Teradata ODBC Connector
- Ability to use the data source to which the Simba Teradata 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.

# Contents

# About the Simba Teradata ODBC Connector

The Simba Teradata ODBC Connector enables Business Intelligence (BI), analytics, and reporting on data that is stored in Teradata Database. 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 Teradata ODBC Connector is available for Microsoft® Windows®, Unix, and macOS platforms.

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

# 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

The Simba Teradata ODBC Connector supports Teradata Database versions 16.20 and later.

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.
- A machine that meets the following system requirements:
  - One of the following operating systems:
    - Windows 11, 10, 8.1, or 7 SP1
    - Windows Server 2022, 2016, 2012 R2, 2012, or 2008 R2 SP1
  - 150 MB of available disk space
  - Visual C++ Redistributable for Visual Studio 2012 installed (with the same bitness as the connector that you are installing).
    You can download the installation packages at https://www.microsoft.com/en-us/download/details.aspx?id=30679.

# 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:

- `Simba Teradata 20.00 32-bit.msi` for 32-bit applications
- `Simba Teradata 20.00 64-bit.msi` for 64-bit applications

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

To install the Simba Teradata ODBC Connector in Windows:

1. Depending on the bitness of your client application, double-click to run **Simba Teradata 20.00 32-bit.msi** or **Simba Teradata 20.00 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 Teradata 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. Open the ODBC Data Source Administrator corresponding to the bitness of the connector that you installed.
2. 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, such as Sisense, load the data using a different user account, and might not be able to detect User DSNs that are created under another user account.

3. Click **Add**.
4. In the Create New Data Source dialog box, select **Simba Teradata ODBC Connector** and then click **Finish**. The Simba Teradata ODBC Connector DSN Setup dialog box opens.
5. In the **Name** field, type a name for your DSN.
6. Optionally, in the **Description** field, type relevant details about the DSN.
7. In the **Name or IP Address** field, type the IP address or host name of the Teradata Database instance.
8. Use the options in the Authentication area to configure authentication for your connection. For more information, see Configuring Authentication in Windows.

> **Note:**
> If the TeraGSS program specifies the appropriate authentication settings for your connection, then you do not need to configure authentication settings in the connector. By default, the connector uses the authentication mechanism that the TeraGSS program specifies in the `tdgssconfigure.xml` file.
>
> Typically, the TeraGSS program specifies TD2 as the authentication mechanism to use.

9. Configure the following optional settings if needed:
   a. In the **Default Database** field, type the name of the database to access by default.
   b. In the **Account String** field, type your account string for accessing the database.
   c. To access additional optional settings, click **Options**. For more information, see Configuring Additional Connector Options in Windows.
10. From the **Session Character Set** drop-down list, select the character set to use for the session.
11. To configure logging behavior for the connector, click **Logging Options**. For more information, see Configuring Logging Options in Windows.
12. To test the connection, click **Test**. If you have not already specified the necessary authentication parameters and credentials during step 8, type those values in the Test Connection dialog box that opens, and then click **OK**. Review the results of the connection test as needed, and then click **OK**.

> **Note:**
> If the connection fails, then confirm that the settings in the Simba Teradata ODBC Driver DSN Setup and Test Connection dialog boxes are correct. Contact your Teradata Database server administrator as needed.

13. To save your settings and close the Simba Teradata ODBC Connector DSN Setup dialog box, click **OK**.
14. To close the ODBC Data Source Administrator, click **OK**.

# Configuring Authentication in Windows

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

- Using Single-Sign On (SSO)
- Using TD2
- Using LDAP
- Using Kerberos
- Using Teradata Negotiating (TDNEGO)
- Using a JSON Web Token (JWT)

- Using Federated Authentication
- Using Federated Authentication
- Using Client Credentials
- Using OIDC Device Authorization Grant
- Using OIDC Token Memory Cache

**Note:**
If you do not specify any authentication settings, then the connector uses the authentication mechanism specified in the `tdgssconfigure.xml` file in the TeraGSS program. This is typically TD2.

# Using Single-Sign On (SSO)

You can configure the connector to authenticate the connection by using Teradata Database credentials that are derived from the user information on your client machine.

To configure SSO 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. From the **Mechanism** drop-down list, select the authentication mechanism that you want the connector to use.
3. Select the **Use Integrated Security** check box.
4. To save your settings and close the dialog box, click **OK**.

# Using TD2

You can configure the connector to use the TD2 protocol to authenticate the connection. For this authentication mechanism, you must provide your user name and password for accessing your Teradata Database instance.

To configure TD2 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. From the **Mechanism** drop-down list, select **TD2**.
3. In the **Username** field, type your Teradata Database user name.
4. In the **Password Wallet String** field, type the Teradata Wallet reference string that is mapped to your Teradata Database password.
5. Optionally, if your database configuration requires you to specify additional parameters for authentication, then in the **Parameter Wallet String** field, type the Teradata Wallet reference string

that is mapped to your authentication parameters.

6. 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. For this authentication mechanism, you do not need to provide a user name and password. The application provides the user name and password.

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. From the **Mechanism** drop-down list, select **LDAP**.
3. Optionally, if your database configuration requires you to specify additional parameters for authentication, then in the **Parameter Wallet String** field, type the Teradata Wallet reference string that is mapped to your authentication parameters.
4. 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. For this authentication mechanism, you do not need to provide a user name and password. The application provides the user name and password.

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. From the **Mechanism** drop-down list, select **KRB5**.
3. Optionally, if your database configuration requires you to specify additional parameters for authentication, then in the **Parameter Wallet String** field, type the Teradata Wallet reference string that is mapped to your authentication parameters.
4. To save your settings and close the dialog box, click **OK**.

# Using Teradata Negotiating (TDNEGO)

You can configure the connector to select the authentication mechanism to use through Teradata Negotiating. Depending on the mechanism that the connector selects as a result of the negotiation process, you might need to provide a user name and password.

To configure TDNEGO 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. From the **Mechanism** drop-down list, select **TDNEGO**.

3. Optionally, if your database configuration requires you to specify additional parameters for authentication, then in the **Parameter Wallet String** field, type the Teradata Wallet reference string that is mapped to your authentication parameters.
4. To save your settings and close the dialog box, click **OK**.

# Using a JSON Web Token (JWT)

To configure JWT 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. From the **Mechanism** drop-down list, select **JWT**.
3. In the **Parameter Wallet String** field, type the Teradata Wallet reference string that is mapped to the token parameter that specifies your JWT.
4. To save your settings and close the dialog box, click **OK**.

# Using Federated Authentication

You can configure the connector to authenticate the connection using Federated Authentication. When a user connects to the data source, the user's credentials are obtained through Keycloak or PingFederate login using an external browser, and the user does not provide a user name or password to the connector.

To configure Federated 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. From the **Mechanism** drop-down list, select **EXTERNALBROWSER**.
3. To save your settings and close the dialog box, click **OK**.

# Using Client Credentials

You can configure the connector to authenticate the connection by using client credentials. This authentication is also known as Bearer Flow.

To configure Client Credentials 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. From the **Mechanism** drop-down list, select **BEARER**.
3. Select the **Mechanism Options** button to configure the following parameters:
   a. In the **JWS_PRIVATE_KEY** field, type the file name of the PEM or JWK file containing the private key.
   b. Optionally, in the **OIDC_TOKEN_CACHE** field, type your tokens to be stored in memory for caching.
   c. Optionally, in the **OIDC_CLIENTID** field, type your OpenID Connect (OIDC) Client ID.

d. Optionally, in the **JWS_CERT** field, type the file name of the X.509 certificate PEM file containing the public key.

e. Optionally, in the **JWS_ALGORITHM** field, specify the JSON Web Signature (JWS) algorithm you want to use.

f. Optionally, in the **OIDC_LOGIN_HINT** field, type your username to login.

g. Optionally, in the **OIDC_PROMPT** field, type your appropriate prompts.

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

# Using OIDC Device Authorization Grant

You can configure the connector to authenticate the connection by using OIDC Device Authorization Grant, also known as OIDC Device Flow.

To configure OIDC Device Authorization Grant 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. From the **Mechanism** drop-down list, select **CODE**.
3. Select the **Mechanism Options** button to configure the following parameters:
   a. Optionally, in the **CODE_APPEND_FILE** field, specify how you want the verification URL and code to display.
   b. Optionally, in the **OIDC_CLIENTID** field, type the Client ID used for Device Code flow.
   c. Optionally, in the **OIDC_SCOPE** field, type the OpenID Connect (OIDC) scope used for Device Code flow.
4. To save your settings and close the dialog box, click **OK**.

# Using OIDC Token Memory Cache

You can configure the connector to reuse credentials and log in multiple times for short-lived connections without refreshing tokens to authenticate the connection. For this authentication mechanism, you must provide your user name and password to the Identity Provider for each and every connection to enable the connector to obtain a new access token and to log on to the database.

To configure OIDC Token Memory Cache  in Windows:

1. To access **DisableTokenCache** option, open the ODBC Data Source Administrator where you created the DSN, select the DSN, and then click **Configure**.
2. Click on **Mechanism** Option.
3. From the **Disable Token Cache** dropdown list, select **Yes**.
4. From the **Mechanism** drop-down list, select **CODE**.
5. Select the **Mechanism** Option and configure the parameters.
6. To save your settings and close the dialog box, click **OK**.

# Configuring Additional Connector Options in Windows

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

To configure additional connector options in Windows:

1. To access additional options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Options**.
2. To return column names instead of column titles when retrieving data, select the **Use Column Names** check box.
3. To use X views so that the connector can only access objects that the specified user owns or controls, select the **Use X View** check box.
4. To disable the HELP database, select the **No HELP DATABASE** check box.
5. To treat the underscore (_) and percent sign (%) characters as normal characters instead of search wildcards, select the **Ignore Search Patterns** check box.
6. To disable the connector SQL parser and pass SQL statements through to the database unchanged, select the **Disable Parsing** check box.
7. To encrypt all communication between the connector and the database, select the **Enable Data Encryption** check box.
8. To return output parameters as a result set, select the **Return Output Parameters As ResultSet** check box.
9. To display decimal symbols based on regional settings, select the **Use Regional Settings for Decimal Symbol** check box.
10. To use extended statement information and enable support for the SQLDescribeParam ODBC API function, select the **Enable Extended Statement Information** check box.
11. To specify the session mode that the connector uses during sessions on the database, from the **Session Mode** drop-down list, select the appropriate mode.
12. To specify the format that the connector uses for DATE values when communicating with the database, from the **Date Time Format** drop-down list, select **AAA** for ANSI format or **IAA** for Integer format.
13. To specify how auto-generated keys are returned for requests that insert data into identity columns, from the **Return Generated Keys** drop-down list, select one of the following methods:
    - **Whole Row**: The entire row is returned.
    - **Identity Column**: Only data from the identity column is returned.
    - **No**: Auto-generated keys are not returned.
14. To specify whether the connector supports Unicode Pass Through (UPT) for Pass Through Characters (PTCs), from the **UPT Mode** drop-down list, select one of the following settings:

- **Notset**: The connector does not do anything to change UPT support.
- **UPTON**: The connector sends a query to the database to enable UPT support.
- **UPTOFF**: The connector sends a query to the database to disable UPT support.

> **Note:**
> For more information about UPT, see "Unicode Pass Through" in the Teradata Database documentation: http://info.teradata.com/htmlpubs/DB_TTU_16_00/index.html#page/General_Reference/B035-1098-160K/ifk1472240714022.html.

15. To enable the creating or updating of User-Defined Functions (UDFs) on the Teradata Database server based on UDFs that are saved in files on the client machine, do the following:
    a. Select the **Enable Client Side UDF Upload** check box.
    b. In the **UDF Upload Path** field, type the full path to the directory where the UDF files are stored. The connector automatically prepends this path to the file names that you specify in the EXTERNAL NAME clauses for CREATE FUNCTION or REPLACE FUNCTION calls.

> **Note:**
> - You have the option of deleting the default value and leaving this field empty. However, we do not recommend setting this option to an empty string, as doing so presents a security risk. For information about the connector behavior and security risk associated with leaving UDF Upload Path empty, see UDF Upload Path.
> - For information about the supported statement syntax, see "CREATE FUNCTION and REPLACE FUNCTION (External Form)" in *SQL Data Definition Language Detailed Topics* from Teradata: https://info.teradata.com/HTMLPubs/DB_TTU_16_00/index.html#page/SQL_Reference%2FB035-1184-160K%2Fbcb1472241301533.html%23.

16. To enable the Redrive feature, so that the connector can attempt to reconnect to the database and resume activities after an interruption has occurred, do the following:
    a. From the **Enable Redrive** drop-down list, select one of the following values:
        - To enable Redrive in the connector, select **Yes**.
        - Or, to configure the connector to determine this setting based on the configuration of the database, select **Default**.

> **Note:**
>
> - In order for the connector to use Redrive, the feature must also be enabled on the database that you are connecting to. For more information, see "Redrive Protection for Queries" in *Teradata Database Administration*: https://docs.teradata.com/reader/B7Lgdw6r3719WUyiCSJcgw/tGnoclQ5P79MZYUu52NDdw.
> - When you connect with Redrive enabled, you can enable or disable the feature on the statement level by using the SQL_ATTR_REDRIVE(13009) statement attribute. For more information, see Redrive Support.

b. In the **Reconnect Count** field, type the maximum number of times that the connector tries to reconnect.

c. In the **Reconnect Interval** field, type the number of seconds that the connector waits between reconnection attempts.

17. To enable the FastExport feature, which improves performance for certain SELECT queries, do the following:
    a. In the **Type** drop-down list, select **FastExport**.
    b. Optionally, in the **Sessions** field, type the number of FastExport data connections that the connector opens to support the performance improvements. Be aware that, at maximum, the connector only opens a number of connections equal to the number of AMPs (Access Module Processors) that are available for your database.

    We recommend that you do not specify a value in the Sessions field. When this property is not set, the number of FastExport connections is determined automatically based on the database settings.

> **Note:**
>
> - In order for the connector to use FastExport, the protocol must also be available on the database that you are connecting to.
> - FastExport is applicable to certain queries only. For more information, see FastExport Support.

18. Optionally, specify SSL connection options. For more information, see Configuring SSL Verification in Windows.
19. Optionally, to automatically close the browser tab opened by Federate authentication, do the following:
    a. In the **EXTERNALBROWSER Tab Timeout** drop-down list, select **Enabled**.
    b. In the **Timeout (seconds)** field, specify the time, in seconds, to wait before the browser closes after authenticating.
20. To configure advanced connector options, click **Advanced**. For more information, see Configuring Advanced Options in Windows.

> **Important:**
> Do not modify the advanced connector options unless your system administrator instructs you to do so. These options are needed in specific scenarios only, and may cause unexpected connector behavior if not configured appropriately.

21. To save your settings and close the Driver Options dialog box, click **OK**.

# Configuring Proxy Server in Windows

You can configure proxy server options to modify the behavior of the connector.

> **Important:**
> Do not modify the proxy server connector options unless your system administrator instructs you to do so. These options are needed in specific scenarios only, and may cause unexpected connector behavior if not configured appropriately.

To configure advanced options in Windows:

1. To access proxy server options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, then click **Options**, then click **Proxy Sever**.
2. In the **HTTP_PROXY** field, specify the host name or IP address of the HTTP proxy server.
3. In the **HTTP_PROXY_USER** field, specify the proxy server username for the HTTP_PROXY server.
4. In the **HTTP_PROXY_PASSWORD** field, specify the proxy server password for the HTTP_ PROXY server.
5. In the **HTTPS_PROXY** field, specify the host name or IP address of the HTTPS proxy server.
6. In the **HTTPS_PROXY_USER** field, specify the proxy server username for the HTTPS_PROXY server.
7. In the **HTTPS_PROXY_PASSWORD** field, specify the proxy server password for the HTTPS_ PROXY server.
8. In the **ALL_PROXY** field, specify the host name or IP address of the proxy server, supports both HTTP and HTTPS proxy.
9. In the **ALL_PROXY_USER** field, specify the proxy server username for the ALL_PROXY server.
10. In the **ALL_PROXY_PASSWORD** field, specify the proxy server password for the ALL_PROXY server.
11. In the **PROXY_BYPASS_HOSTS** field, specify a comma separated list of host name, domain, and IP address.

12. To save your settings and close the Advanced Options dialog box, click **OK**.

> **Note:** The connector can now read from `HTTP_PROXY`, `HTTPS_PROXY`, and `ALL_PROXY` settings based on the URI protocol from the Identity Provider.
>
> - If the URI is "`http://`", `HTTP_PROXY` is used.
> - If the URI is "`https://`", `HTTPS_PROXY` is used.
> - If both are unavailable , `ALL_PROXY` is used.

# Configuring Advanced Options in Windows

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

> **Important:**
> Do not modify the advanced connector options unless your system administrator instructs you to do so. These options are needed in specific scenarios only, and may cause unexpected connector behavior if not configured appropriately.

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**, then click **Options**, then click **Advanced**.
2. In the **Maximum Response Buffer** field, specify the maximum size of the response buffer for SQL requests, in kilobytes.
3. In the **TDMST Port Number** field, specify the number of the port used to access Teradata Database.
4. In the **HTTPS Port Number** field, specify the number of the port used to access Teradata Database when using TLS.
5. In the **Translation DLL Name** field, specify the `.dll` file that contains functions for translating all the data that is transferred between the Teradata server and the connector. This `.dll` file is used for translation if local character sets are not supported by Teradata Database or the connector.
6. In the **Translation Option** field, specify the options used by the Translation DLL file. The required options may vary depending on the Translation DLL file being used.
7. In the **Login Timeout** field, specify the number of seconds to wait for a response when logging in to the database.
8. To enable the print option when creating stored procedures, from the **Procedure With Print Stmt** drop-down list, select **P**.
9. To enable the SPL option when creating stored procedures, from the **Procedure With SPL Source** drop-down list, select **Y**.
10. In the **Data Source DNS Entries** field, specify how the connector determines which DNS entry to use by doing one of the following:

- To resolve DNS entries dynamically, leave the field empty.
- Or, to use DNS lookup, type **0**.
- Or, to specify a number of DNS entries to use in a round-robin fashion, type the number of entries.

11. To use the TCP_NODELAY setting, select the **Use TCP_NODELAY** check box.
12. To specify NULL for the Catalog Name parameter in all Catalog API functions, select the **Use NULL For Catalog Name** check box.
13. To have the connector request the next response message while it is processing the current response message, select the **Enable Read Ahead** check box.
14. To retry socket system calls at the connector level instead of the application level, select the **Retry System Calls (EINTR)** check box.
15. To optimize retrieval for Large Object (LOB) data that meets specified size requirements, enable Smart LOB (SLOB) Mode by doing the following. For detailed information about the supported LOB retrieval modes, see LOB Retrieval Modes.
    a. In the **Max Single LOB Bytes** field, type the maximum size of the LOBs (in bytes) that the connector can retrieve using SLOB Mode. LOBs that exceed this size are retrieved using Deferred Mode instead.
    b. In the **Max Total LOB Bytes Per Row** field, type the maximum size of LOB data per row (in bytes) that the connector can retrieve using SLOB Mode. If the total amount of LOB data being retrieved from a row exceeds this size, then after using SLOB Mode to retrieve LOBs up to this size limit, the connector uses Deferred Mode to retrieve the remaining LOBs from that row.
    c. If you are retrieving LOB data from columns in sequential order, select the **Use Sequential Retrieval Only** check box.

    **Important:**
    If you enable this option but then retrieve LOB data from columns in a non-sequential order, connector performance may decrease. In this scenario, the connector discards the LOBs that are returned through SLOB Mode and must then retrieve them all again using Deferred Mode.

16. To enable compatibility with applications that use Microsoft Access Jet databases by using DATE data in TIMESTAMP parameters, select the **Use DATE Data For TIMESTAMP Parameters** check box.
17. To provide backwards compatibility for ODBC 2.x applications that use noncompliant search patterns, select the **Enable Custom Catalog Mode For 2.x Applications** check box.
18. To return an empty string in the CREATE_PARAMS column when you call SQLGetTypeInfo for SQL_TIMESTAMP data, select the **Return Empty String In CREATE_PARAMS Column For SQL_TIMESTAMP** check box.
19. To return a hard-coded value as the maximum length of SQL_CHAR and SQL_VARCHAR columns, select the **Return Max CHAR/VARCHAR Length As 32K** check box.

**Note:**

- Enabling this option prevents the returned column size from causing numeric overflows in Microsoft Access.
- The hard-coded value is either 32000 or 64000, depending on the setting specified for the Session Character Set connector option.

20. To save your settings and close the Advanced Options dialog box, click **OK**.

# Configuring SSL Verification in Windows

If you are connecting to a Teradata server that has Secure Sockets Layer (SSL) enabled, you can configure the connector to connect to an SSL-enabled socket.

To configure SSL verification in Windows:

1. To access SSL options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Options**.
2. From the SSL Mode drop-down list, select the TLS mode:
   - **Allow**: The connector connects to the data store using the TDMST port if it is enabled; if not, the connector uses the HTTPS port.

     If both ports are enabled, the connector connects to the data store using the TDMST port. If this is unsuccessful, the connector returns an error.
   - **Disable**: The connector only connects to the data store using the TDMST port.
   - **Prefer**: The connector connects to the data store using the HTTPS port if it is enabled; if not, the connector uses the TDMST port.

     If both ports are enabled, the connector connects to the data store using the HTTPS port. If this is unsuccessful, the connector returns an error.
   - **Require**: The connector only connects to the data store using the HTTPS port.
   - **Verify-CA**: The connector only connects to the data store using the HTTPS port. In addition, the connector verifies the server CA certificate against the configured CA certificates.
   - **Verify-Full**: The connector only connects to the data store using the HTTPS port. In addition, the connector verifies the server CA certificate against the configured CA certificates, and performs additional host name identity verification.

     **Note:**
     For a detailed explanation of the deterministic behavior of the Allow and Prefer modes, see Deterministic Behavior for Prefer and Allow SSL Modes.

3. If the SSL Mode is set to Verify-CA or Verify-Full:
   - To verify the server using the trusted CA certificates from a specific directory, in the **SSL CA Path** field, specify the full path to the directory.
   - Or, to verify the server using use a specific `.pem` file, in the **SSL CA File Name** field, specify the full path to the file including the file name.
4. Optionally, to specify an HTTPS port other than the default port 443, do the following:
   a. Click **Advanced**. The Advanced Options dialog box opens.
   b. In the **HTTPS Port Number** field, specify the number of the port used to access Teradata Database when using SSL.
   c. Click **OK** to save your settings and close the Advanced Options dialog box.
5. To save your settings and close the Additional Options dialog box, click **OK**.

# Configuring Logging Options in Windows

To help troubleshoot issues, you can enable logging. In addition to functionality provided in the Simba Simba Teradata 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 Teradata ODBC Connector, so make sure to disable the feature after you are done using it.

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.

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. To log error events in the Event Viewer of the Teradata server, select the **Enable Event Tracing for Windows** check box.
7. Click **OK**.
8. Restart your ODBC application to make sure that the new settings take effect.

The Simba Teradata ODBC Connector produces the following log files at the location you specify in the Log Path field, where *[UserName]* and *[ProcessID]* are the user name and process ID associated with the connection that is being logged, and *[Number]* is a number that identifies each log file:

- A `[UserName]_[ProcessID]_simbateradataodbcdriver.log` file that logs connector activity that is not specific to a connection.
- A `[UserName]_[ProcessID]_simbateradataodbcdriver_connection_[Number].log` file for each connection made to the database.

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.

# Verifying the Driver Version Number in Windows

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

To verify the driver version number in Windows:

1. Open the ODBC Data Source Administrator corresponding to the bitness of the driver that you installed.
2. Click the **Drivers** tab and then find the Simba Teradata ODBC Connector in the list of ODBC drivers 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

The Simba Teradata ODBC Connector supports Teradata Database versions 16.10 and later.

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:

- One of the following macOS versions:
  - macOS 10.14 (Universal Binary – Intel and ARM support)
  - macOS 10.15 (Universal Binary – Intel and ARM support)
  - macOS 11 (Universal Binary – Intel and ARM support)
  - macOS 12 (Universal Binary – Intel and ARM support)
- 100MB of available disk space
- One of the following ODBC driver managers installed:
  - iODBC 3.52.9 or later
  - unixODBC 2.2.14 or later

  **Note:** .

# 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 Teradata ODBC Connector is available for macOS as a `.dmg` file named `Simba Teradata 20.00.dmg`. The connector supports both 32- and 64-bit client applications.

To install the Simba Teradata ODBC Connector in macOS:

1. Double-click **Simba Teradata 20.00.dmg** to mount the disk image.
2. Double-click **SimbaTeradata20.00.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/teradata` 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/teradata/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

   Next, configure the environment variables on your machine to make sure that the ODBC Driver manager can work with the connector. For more information,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 Teradata 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.teradataodbc
```

The command returns information about the Simba Teradata 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.

# Unix System Requirements

The Simba Teradata ODBC Connector supports Teradata Database versions 16.10, 16.20, 17.00, 17.10, and 17.20.

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:

- One of the following distributions:
  - Red Hat® Enterprise Linux® (RHEL) 6 or 7 or 8 or 8.5 or 9
  - CentOS 6 or 7 or 8
  - SUSE Linux Enterprise Server (SLES) 11 or 12 or 15
  - Ubuntu 16.04, 18.04, 20.04, or 22.04
  - Oracle Enterprise Linux 8
  - AIX 7.1 or 7.2
  - Solaris Sparc 11.3
  - Solaris Opteron 11.3

  > **Note:**
  >
  > For AIX and HP-UX, a special build of the connector is required. For more information, contact the Sales team: https://www.simba.com/contact-us/.

- 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

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*.

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.

You can install both the 32-bit and 64-bit versions of the connector on the same machine.

To install the Simba Teradata 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 Unix 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]*

   The Simba Teradata ODBC Connector files are installed in the `/opt/simba/teradata` directory.

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 on Unix

If you need to verify the version of the Simba Teradata ODBC Connector that is installed on your Unix 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 on Unix using the command-line interface:

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

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

To verify the connector version number on Unix 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/teradata/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 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 Teradata 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 on 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 on 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.

## Unix

If you are using an Unix 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 Unix shell documentation.

## Troubleshooting

When you attempt to connect through the connector on an Unix machine, you may encounter the following error message:

SQLDriverConnect = [Simba][ODBC] (11560) Unable to locate SQLGetPrivateProfileString function. (11560)

This issue may occur when the name of the library file for the driver manager is different from the default. To resolve this issue, do the following:

1. Confirm the name of the library file that is used by your driver manager.
2. In a text editor, open the `simba.teradataodbc.ini` file (located in *`[InstallDir]`*`/lib` by default).
3. Add the following line to the end of the file, where *[DMLibFile]* is the name of the library file:

   ODBCInstLib=*[DMLibFile]*
4. Save the `simba.teradataodbc.ini` file.

# 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.teradataodbc.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 SIMBAODBCINI to the full path and file name of the `simba.teradataodbc.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 SIMBAODBCINI to the full path and file name of the `simba.teradataodbc.ini` file.

For example, if your `odbc.ini` and `odbcinst.ini` files are located in `/usr/local/odbc` and your `simba.teradataodbc.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 SIMBAODBCINI=/etc/simba.teradataodbc.ini
```

**For unixODBC:**

```
export ODBCINI=/usr/local/odbc/odbc.ini
export ODBCSYSINI=/usr/local/odbc
export SIMBAODBCINI=/etc/simba.teradataodbc.ini
```

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

1. If the SIMBAODBCINI 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.teradataodbc.ini`.
3. The connector searches the current working directory of the application for a file named `simba.teradataodbc.ini`.
4. The connector searches the home directory for a hidden file named `.simba.teradataodbc.ini` (prefixed with a period).
5. The connector searches the `/etc` directory for a file named `simba.teradataodbc.ini`.

# Configuring ODBC Connections in Non-Windows Machine

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

- Creating a Data Source Name on a Non-Windows Machine
- Configuring a DSN-less Connection on a Non-Windows Machine
- Configuring Authentication 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 on 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=Teradata ODBC Driver

   As another example, for a 32-bit connector on an Unix machine:

   [ODBC Data Sources]

   Sample DSN=Teradata ODBC Driver 32-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/teradata/lib/tdataodbc_sbu.dylib

   As another example, for a 32-bit connector on an Unix machine:

   Driver=/opt/simba/teradata/lib/32/tdataodbc_sb32.so

b. Set the `DBCName` property to the IP address or host name of the Teradata Database instance.

   For example:

   DBCName=192.168.222.160

c. Configure authentication for your connection by specifying the authentication mechanism and your credentials as needed. For more information, see Configuring Authentication on a Non-Windows Machine.

> **Note:**
> If the TeraGSS program specifies the appropriate authentication settings for your connection, then you do not need to configure authentication settings in the connector. By default, the connector uses the authentication mechanism that the TeraGSS program specifies in the `tdgssconfigure.xml` file.
>
> Typically, the TeraGSS program specifies TD2 as the authentication mechanism to use.

d. Optionally, set additional key-value pairs as needed to specify other connection settings. For detailed information about each connection property, see Connector Configuration Properties.

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

   For example, the following is an `odbc.ini` configuration file for macOS containing a DSN that connects to Teradata:

   [ODBC Data Sources]

   Sample DSN=Teradata ODBC Driver

   [Sample DSN]

   Driver=/Library/simba/teradata/lib/tdataodbc_sbu.dylib

   DBCName=192.168.222.160

   MechanismName=KRB5

   As another example, the following is an `odbc.ini` configuration file for a 32-bit connector on an Unix machine, containing a DSN that connects to Teradata:

   [ODBC Data Sources]

   Sample DSN=Teradata ODBC Driver 32-bit

   [Sample DSN]

   Driver=/opt/simba/teradata/lib/32/tdataodbc_sb32.so

DBCName=192.168.222.160

MechanismName=KRB5

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, on a macOS machine:

   [ODBC Drivers]

   Teradata ODBC Driver=Installed

   As another example, for a 32-bit connector on an Unix machine:

   [ODBC Drivers]

   Teradata ODBC Driver 32-bit=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/teradata/lib/tdataodbc_sbu.dylib

      As another example, for a 32-bit connector on an Unix machine:

      Driver=/opt/simba/teradata/lib/32/tdataodbc_sb32.so

   b. Optionally, set the `Description` property to a description of the connector.

      For example:

      Description=Teradata ODBC Driver

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]

Teradata ODBC Driver=Installed

[Teradata ODBC Driver]

Driver=/Library/simba/teradata/lib/tdataodbc_sbu.dylib

Description=Teradata ODBC Driver

For example, the following is an `odbcinst.ini` configuration file for both the 32- and 64-bit connectors on Unix:

[ODBC Drivers]

Teradata ODBC Driver 32-bit=Installed

Teradata ODBC Driver 64-bit=Installed

[Teradata ODBC Driver 32-bit]

Driver=/opt/simba/teradata/lib/32/tdataodbc_sb32.so

Description=Teradata ODBC Driver (32-bit)

[Teradata ODBC Driver 64-bit]

Driver=/opt/simba/teradata/lib/64/tdataodbc_sb64.so

Description=Teradata ODBC Driver (64-bit)

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 authentication, see Configuring Authentication on a Non-Windows Machine.

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

# Configuring Authentication on a Non-Windows Machine

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

- Using Single-Sign On (SSO)
- Using TD2
- Using LDAP
- Using Kerberos
- Using Teradata Negotiating (TDNEGO)
- Using a JSON Web Token (JWT)
- Using Federated Authentication (macOS only)
- Using Client Credentials
- Using OIDC Device Authorization Grant
- Using OIDC Token Memory Cache

**Note:**
If you do not specify any authentication settings, then the connector uses the authentication mechanism specified in the `tdgssconfigure.xml` file in the TeraGSS program. This is typically TD2.

# Using Single-Sign On (SSO)

You can configure the connector to authenticate the connection by using Teradata Database credentials that are derived from the user information on your client machine.

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.

To configure SSO on a non-Windows machine:

1. Set the `MechanismName` property to `TD2`.
2. Set the `UseIntegratedSecurity` property to `1`.

# Using TD2

You can configure the connector to use the TD2 protocol to authenticate the connection. For this authentication mechanism, you must provide your user name and password for accessing your Teradata Database instance.

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. For examples of connection strings, see Using a Connection String.

To configure TD2 authentication on a non-Windows machine:

1. Set the `MechanismName` property to `TD2`.
2. Set the `UID` property to your Teradata Database user name.
3. Set the `WalletString` property to the Teradata Wallet reference string that is mapped to your Teradata Database password. Use the following format, where *[ReferenceString]* is your reference string:

   `WalletString`=$tdwallet(*[ReferenceString]*)
4. Optionally, if your database configuration requires you to specify additional parameters for authentication, set the `AuthenticationParameter` property to the Teradata Wallet reference string that is mapped to your authentication parameters. Use the following format, where *[ReferenceString]* is your reference string:

   AuthenticationParameter=$tdwallet(*[ReferenceString]*)

# Using LDAP

You can configure the connector to use the LDAP protocol to authenticate the connection. For this authentication mechanism, you do not need to provide a user name and password. The application provides the user name and password.

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. For examples of connection strings, see Using a Connection String.

To configure LDAP authentication on a non-Windows machine:

1. Set the `MechanismName` property to `LDAP`.
2. Optionally, if your database configuration requires you to specify additional parameters for authentication, set the `AuthenticationParameter` property to the Teradata Wallet reference string that is mapped to your authentication parameters. Use the following format, where *[ReferenceString]* is your reference string:

   AuthenticationParameter=$tdwallet(*[ReferenceString]*)

# Using Kerberos

You can configure the connector to use the Kerberos protocol to authenticate the connection. For this authentication mechanism, you do not need to provide a user name and password. The application provides the user name and password.

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. For examples of connection strings, see Using a Connection String.

To configure Kerberos authentication on a non-Windows machine:

1. Set the `MechanismName` property to `KRB5`.
2. Optionally, if your database configuration requires you to specify additional parameters for authentication, set the `AuthenticationParameter` property to the Teradata Wallet reference string that is mapped to your authentication parameters. Use the following format, where *[ReferenceString]* is your reference string:

   AuthenticationParameter=$tdwallet(*[ReferenceString]*)

# Using Teradata Negotiating (TDNEGO)

You can configure the connector to select the authentication mechanism to use through Teradata Negotiating. Depending on the mechanism that the connector selects as a result of the negotiation process, you might need to provide a user name and password.

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. For examples of connection strings, see Using a Connection String.

To configure TDNEGO authentication on a non-Windows machine:

1. Set the `MechanismName` property to `TDNEGO`.
2. Optionally, if your database configuration requires you to specify additional parameters for authentication, set the `AuthenticationParameter` property to the Teradata Wallet reference string that is mapped to your authentication parameters. Use the following format, where *[ReferenceString]* is your reference string:

   AuthenticationParameter=$tdwallet(*[ReferenceString]*)

# Using a JSON Web Token (JWT)

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. For examples of connection strings, see Using a Connection String.

To configure JWT authentication on a non-Windows machine:

1. Set the `MechanismName` property to `JWT`.
2. Set the `AuthenticationParameter` property to the Teradata Wallet reference string that is mapped to the token parameter that specifies your JWT. Use the following format, where *[ReferenceString]* is your reference string:

   AuthenticationParameter=$tdwallet(*[ReferenceString]*)

# Using Federated Authentication

In macOS, you can configure the connector to authenticate the connection using Federated Authentication. When a user connects to the data source, the user's credentials are obtained through Keycloak or PingFederate login using an external browser, and the user does not provide a user name or password to the connector.

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.

To configure Federated Authentication  in macOS:

- Set the `MechanismName` property to `EXTERNALBROWSER`.

# Using Client Credentials

You can configure the connector to authenticate the connection by using client credentials. This authentication is also known as Bearer Flow.

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. For examples of connection strings, see Using a Connection String.

To configure Client Credentials authentication on a non-Windows machine:

1. Set the `MechanismName` property to `BEARER`.
2. Set the `JWS_PRIVATE_KEY` property to the the file name of the PEM or JWK file containing the private key.
3. Optionally, set the `OIDC_TOKEN_CACHE` property to your tokens to be stored in memory for caching.
4. Optionally, set the `OIDC_CLIENTID` property to your OpenID Connect (OIDC) Client ID.
5. Optionally, set the `JWS_CERT` property to the file name of the X.509 certificate PEM file containing the public key.
6. Optionally, set the `JWS_ALGORITHM` property to the JSON Web Signature (JWS) algorithm you want to use.
7. Optionally, set the `OIDC_LOGIN_HINT` property to your username to login.
8. Optionally, set the `OIDC_PROMPT` property to your appropriate prompts.

# Using OIDC Device Authorization Grant

You can configure the connector to authenticate the connection by using OIDC Device Authorization Grant, also known as OIDC Device Flow.

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. For examples of connection strings, see Using a Connection String.

To configure OIDC Device Authorization Grant authentication on a non-Windows machine:

1. Set the `MechanismName` property to `CODE`.
2. Optionally, set the `CODE_APPEND_FILE` property to specify how you want the verification URL and code to display.
3. Optionally, set the `OIDC_CLIENTID` property to the Client ID used for Device Code flow.
4. Optionally, set the `OIDC_SCOPE` property to the OpenID Connect (OIDC) scope used for Device Code flow.

# Using OIDC Token Memory Cache

You can configure the connector to reuse credentials and log in multiple times for short-lived connections without refreshing tokens to authenticate the connection. For this authentication mechanism, you must provide your user name and password to the Identity Provider for each and every connection to enable the connector to obtain a new access token and to log on to the database.

To configure OIDC Token Memory Cache  on non Windows:

- Set the `DisableTokenCache` property to `Yes` or `No`.

# Configuring SSL Verification on a Non-Windows Machine

If you are connecting to a Teradata server that has Secure Sockets Layer (SSL) enabled, you can configure the connector to connect to an SSL-enabled socket.

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.

To configure SSL verification  on a Non-Windows Machine:

1. Set the `SSLMode` property to one of the following:
   - `Allow`: The connector connects to the data store using the TDMST port if it is enabled; if not, the connector uses the HTTPS port.
   - If both ports are enabled, the connector connects to the data store using the TDMST port. If this is unsuccessful, the connector returns an error.
   - `Disable`: The connector only connects to the data store using the TDMST port.
   - `Prefer`: The connector connects to the data store using the HTTPS port if it is enabled; if not, the connector uses the TDMST port.
   - If both ports are enabled, the connector connects to the data store using the HTTPS port. If this is unsuccessful, the connector returns an error.
   - `Require`: The connector only connects to the data store using the HTTPS port.
   - `Verify-CA`: The connector only connects to the data store using the HTTPS port. In addition, the connector verifies the server CA certificate against the configured CA certificates.
   - `Verify-Full`: The connector only connects to the data store using the HTTPS port. In addition, the connector verifies the server CA certificate against the configured CA certificates, and performs additional host name identity verification.

     **Note:**
     For a detailed explanation of the deterministic behavior of the Allow and Prefer modes, see Deterministic Behavior for Prefer and Allow SSL Modes.

2. If `SSLMode` is set to `Verify-CA` or `Verify-Full`:
   - To verify the server using the trusted CA certificates from a specific directory, set `SSLCAPath` to the full path to the directory.
   - Or, to verify the server using use a specific `.pem` file, set `SSLCA` to the full path to the file including the file name.
3. Optionally, to specify an HTTPS port other than the default port 443, set `HTTPS_PORT` to the number of the port used to access Teradata Database when using SSL.

# 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.

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.teradataodbc.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. Save the `simba.teradataodbc.ini` configuration file.
6. Restart your ODBC application to make sure that the new settings take effect.

The Simba Teradata ODBC Connector produces the following log files at the location you specify in the Log Path field, where *[UserName]* and *[ProcessID]* are the user name and process ID associated with the connection that is being logged, and *[Number]* is a number that identifies each log file:

- A *[UserName]*_*[ProcessID]*_`simbateradataodbcconnector.log` file that logs connector activity that is not specific to a connection.
- A *[UserName]*_*[ProcessID]*_`simbateradataodbcconnector_connection_`*[Number]*`.log` file for each connection made to the database.

**To disable logging on a non-Windows machine:**

1. Set the `LogLevel` key to `0`.
2. Save the `simba.teradataodbc.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

To write a connection string that uses a DSN, set the `DSN` key to the name of your DSN. As an alternative, you can set the `DataSourceName` key, which is synonymous with the `DSN` key.

The following are examples of connection strings that use a DSN:

DSN=MyDSNForTeradata

DataSourceName=MyDSNForTeradata

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.

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

- *[AuthenticationMechanism]* is the mechanism that the connector uses to authenticate the connection to the database. For information about the supported settings, see Mechanism.
- *[JWT_Token]* is the JSON web token that you obtained from the UDA User Service.
- *[Server]* is the IP address or host name of the Teradata Database instance to which you are connecting.
- *[YourUserName]* is the user name that you use to access the database.
- *[YourPassword]* is the password corresponding to your user name.

## Connecting to a Teradata Database Instance Using Single Sign-On

The following is the format of a DSN-less connection string that connects to the database using Single Sign-On (SSO):

Driver=Simba Teradata ODBC Connector;DBCName=*[Server]*;MechanismName=*[AuthenticationMechanism]*;UseIntegratedSecurity=1;

**Note:**
`MechanismName` is optional. If this option is not set, then the connector uses the authentication mechanism that the TeraGSS program specifies in the `tdgssconfigure.xml` file.

For example:

Driver=Simba Teradata ODBC Connector;DBCName=192.168.222.160;MechanismName=TD2;UseIntegratedSecurity=1;

# Connecting to a Teradata Database Instance Using TD2

The following is the format of a DSN-less connection string that connects to the database using the TD2 protocol:

Driver=Simba Teradata ODBC Connector;DBCName=*[Server]*;MechanismName=TD2;UID=*[YourUserName]*;Password=*[YourPassword]*;

For example:

Driver=Simba Teradata ODBC Connector;DBCName=192.168.222.160;MechanismName=TD2;UID=jsmith;Password=simba123;

Alternatively, you can provide a Teradata Wallet reference string instead of a password.

For example:

Driver=Simba Teradata ODBC Connector;DBCName=192.168.222.160;MechanismName=TD2;UID=jsmith;WalletString=$tdwallet(jsmith_wallet_string);EnableWallet=1;

**Note:**
The Teradata Wallet utility must be installed and configured before you can connect using a reference string.

# Connecting to a Teradata Database Instance Using LDAP, Kerberos, or TDNEGO

The following is the format of a DSN-less connection string that connects to the database using the LDAP, Kerberos, or TDNEGO protocol. For LDAP and Kerberos, you do not need to set the `UID` and `Password` properties, because the connector obtains these credentials from the application. For TDNEGO, depending on the actual mechanism that the connector selects as a result of the negotiation process, you might need to set `UID` and `Password` as shown in the example above for TD2.

Driver=Simba Teradata ODBC Connector;DBCName=*[Server]*;MechanismName=*[AuthenticationMechanism]*;

For example, to use LDAP:

Driver=Simba Teradata ODBC Connector;DBCName=192.168.222.160;MechanismName=LDAP;

# Connecting to a Teradata Database Instance Using a JSON Web Token

The following is the format of a DSN-less connection string that connects to the database using a JSON web token (JWT):

Driver=Simba Teradata ODBC Connector;DBCName=*[Server]*;MechanismName=JWT;AuthenticationParameter={token=*[JWT_Token]*};

For example:

Driver=Simba Teradata ODBC Driver;DBCName=192.168.222.160;MechanismName=JWT;AuthenticationParameter={token=zio5YOBZ.nExFB6lm.SOwvlWy2};

# Features

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

- Data Types
- Security and Authentication
- OIDC Token Memory Cache
- TLS/SSL Encryption
- Data Encryption
- Teradata Wallet
- Redrive Support
- FastExport Support
- LOB Retrieval Modes
- Scalar Function Support
- Special Query Syntax

# Data Types

The Simba Teradata ODBC Connector supports two-way mapping between Teradata SQL types and many common ODBC SQL data types.

The tables below list the supported data types and their mappings. The first table lists Teradata SQL types that are mapped to standard ODBC SQL data types, while the second table lists those that are mapped to custom SQL types.

**Note:**
As indicated below, some Teradata SQL types may be returned differently depending on the character set that is specified in the Session Character Set option (the `CharacterSet` property). For more information, see Session Character Set.

Additionally, some data types are returned differently depending on whether the `IntervalPeriodTypesToString` property is enabled. For more information, see IntervalPeriodTypesToString.

| Teradata SQL Type | ODBC SQL Type |
|---|---|
| BIGINT | SQL_BIGINT |
| BLOB | SQL_LONGVARBINARY |
| BYTE | SQL_BINARY |
| BYTEINT | SQL_TINYINT |

| Teradata SQL Type | ODBC SQL Type |
|---|---|
| CHAR<br>CHARACTER | SQL_CHAR when using a non-Unicode character set.<br>SQL_WCHAR when using a Unicode character set. |
| CLOB | SQL_LONGVARCHAR when using a non-Unicode character set.<br>SQL_WLONGVARCHAR when using a Unicode character set. |
| DATE | SQL_TYPE_DATE |
| DECIMAL | SQL_DECIMAL |
| FLOAT | SQL_DOUBLE |
| NUMBER | See "FIXED_NUMBER" and "FLOATING_NUMBER" in the custom SQL types table below.<br>For detailed information about the handling of the Number data type in Teradata Database, see "Number Data Types" in the *ODBC Driver for Teradata User Guide*. |
| REAL | SQL_REAL |
| INT<br>INTEGER | SQL_INTEGER |
| INTERVAL DAY | SQL_INTERVAL_DAY by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL DAY TO HOUR | SQL_INTERVAL_DAY_TO_HOUR by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL DAY TO MINUTE | SQL_INTERVAL_DAY_TO_MINUTE by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL DAY TO SECOND | SQL_INTERVAL_DAY_TO_SECOND by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL HOUR | SQL_INTERVAL_HOUR by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL HOUR TO MINUTE | SQL_INTERVAL_HOUR_TO_MINUTE by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL HOUR TO SECOND | SQL_INTERVAL_HOUR_TO_SECOND by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |

| Teradata SQL Type | ODBC SQL Type |
|---|---|
| INTERVAL MINUTE | SQL_INTERVAL_MINUTE by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL MINUTE TO SECOND | SQL_INTERVAL_MINUTE_TO_SECOND by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL MONTH | SQL_INTERVAL_MONTH by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL SECOND | SQL_INTERVAL_SECOND by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL YEAR | SQL_INTERVAL_YEAR by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| INTERVAL YEAR TO MONTH | SQL_INTERVAL_YEAR_TO_MONTH by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| SMALLINT | SQL_SMALLINT |
| TIME | SQL_TYPE_TIME |
| TIME WITH TIME ZONE | SQL_TYPE_TIME by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| TIMESTAMP | SQL_TYPE_TIMESTAMP |
| TIMESTAMP WITH TIME ZONE | SQL_TYPE_TIMESTAMP<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |
| VARBYTE | SQL_VARBINARY |
| VARCHAR | SQL_VARCHAR when using a non-Unicode character set.<br>SQL_WVARCHAR when using a Unicode character set. |

The following table lists Teradata SQL types that are mapped to custom SQL types.

| Teradata SQL Type | Custom SQL Types |
|---|---|
| | |
| DATASET STORAGE FORMAT AVRO<br>This data type is used for both Avro and CSV data. For more information, see | SQL_TD_DATASET_AVRO (18006) for Avro data.<br>SQL_TD_DATASET_CSV (18007) for CSV data when using a non-Unicode character set. |

| Teradata SQL Type | Custom SQL Types |
|---|---|
| "DATASET Data Type" in the *ODBC Driver for Teradata User Guide*. | SQL_TD_DATASET_WCSV (18008) for CSV data when using a Unicode character set. |
| FIXED_NUMBER | SQL_TD_FIXED_NUMBER (18001) == SQL_DECIMAL |
| FLOATING_NUMBER | SQL_TD_FLOATING_NUMBER (18002) == SQL_DOUBLE |
| JSON | SQL_TD_JSON (18004) when using a non-Unicode character set.<br>SQL_TD_WJSON (18005) when using a Unicode character set. |
| PERIOD(DATE) | SQL_PERIOD_DATE (-1049) |
| PERIOD(TIME)<br>PERIOD(TIME(n)) | SQL_PERIOD_TIME (-1048) |
| PERIOD(TIME WITH TIME ZONE)<br>PERIOD(TIME(n) WITH TIME ZONE) | SQL_PERIOD_TIME_WITH_TIME_ZONE (-1047) |
| PERIOD(TIMESTAMP)<br>PERIOD(TIMESTAMP(n)) | SQL_PERIOD_TIMESTAMP (-1046) |
| PERIOD(TIMESTAMP WITH TIME ZONE)<br>PERIOD(TIMESTAMP(n) WITH TIME ZONE) | SQL_PERIOD_TIMESTAMP_WITH_TIME_ZONE (-1045) |
| XML | SQL_TD_XML (18003) by default.<br>SQL_WLONGVARCHAR if the `IntervalPeriodTypesToString` property is enabled. |

# Security and Authentication

To protect data from unauthorized access, Teradata data stores require authentication for access. To access your data, you must configure the connector to pass in your credentials and authenticate the connection. The Simba Teradata ODBC Connector supports a number of methods for authenticating connections:

- TD2
- Kerberos (KRB5)
- LDAP
- JSON Web Token (JWT)
- Teradata Negotiating (TDNEGO)
- Single Sign-On (SSO), including SSO through TDNEGO
- Federated Authentication (Windows and macOS only)

Configure authentication for your connection by selecting an authentication mechanism and then specifying the appropriate credentials in the DSN or connection string. In some cases, such as when you use LDAP or KRB5, you do not need to specify any credentials because the connector uses the credentials from the application. For detailed configuration instructions, see Configuring Authentication in Windows or Configuring Authentication on a Non-Windows Machine.

When configuring authentication, you have the option of providing Teradata Wallet reference strings instead of specifying your password or additional authentication parameters directly in the connection information. For security reasons, the connector does not allow password values or authentication parameters to be saved in DSNs. You can only save Teradata Wallet reference strings that are mapped to the password or authentication parameters. For more information about the Teradata Wallet utility, see Teradata Wallet.

# OIDC Token Memory Cache

The Simba Teradata ODBC Connector describes the memory storage of industry standard OAuth 2.0 OIDC access and refresh tokens. For more details, please refer to https://www.oauth.com/.

In the original implementation of client authentication in the Simba Teradata ODBC Connector (earlier to `20.00.00.019`), *External Browser*, *OIDC Bearer Flow*, and *OIDC Device Code Flow* authentications use OIDC access tokens to log on to the database. The user needs to provide credentials (user name and password) to the Identity Provider for each and every connection to enable the connector to obtain a new access token and to log on to the database.

In connector `20.00.00.019` and later a new feature to store access tokens in memory for re-use in subsequent connections has been implemented.

When an application logs into the database for the first time, the user provides credentials to an Identity Provider. If successful, the Identity Provider gives the connector an access token and a refresh token. The access token is used to log into the database, and the refresh token is used to get a new access token when needed. Both tokens are stored securely in memory. As long as the access token has not expired, the user does not need to enter credentials again for future logins.

When the application tries to connect to the database after the initial login, the connector will fetch an access token from the token memory cache. If the token is still valid, it will use this token to log in. If the access token has expired but the refresh token is still valid, the connector will use the refresh token to get a new access token from the Identity Provider. If both tokens have expired, the user will need to provide credentials to the Identity Provider to get new access and refresh tokens.

Tokens in the memory cache are periodically checked (every `60` seconds, though this may change in future connector updates) to see if they are nearing expiration. If they are, the refresh token is sent to the Identity Provider to get new access and refresh tokens. The expiration time and the number of times access tokens can be refreshed are set by the Identity Provider's policies and cannot be changed by the connector. Expired tokens are removed from the token memory cache.

The lifetime of the token memory cache is determined by the ODBC driver manager:

- For UNIX (Linux, OSX, AIX) applications, the token memory cache has the lifetime of the application process. It is created on the first connection attempt and destroyed when the application process exits.

- For Windows, the token memory cache has the lifetime of an ODBC environment, from the first SQLConnect(...), SQLBrowserConnect(...), or SQLDriverConnect(...) call to the

SQLFreeHandle(SQL_HANDLE_ENV, ...) call. (ODBC allows multiple connections within each environment.)

The Token Memory Cache feature is `ON` by default. The Teradata DSN configuration option `DisableTokenCache` can be used to enable (`No`, the default) or disable (`Yes`) this feature. If `DisableTokenCache=Yes` then the connector will not use token memory cache and it will behave the same as connectors prior to `20.00.00.019`.

# TLS/SSL Encryption

The Simba Teradata ODBC Connector supports TLS v1.2 encryption. For detailed configuration instructions, see Configuring SSL Verification in Windows or Configuring SSL Verification on a Non-Windows Machine.

**Note:**
In this documentation, "SSL" refers to both TLS (Transport Layer Security) and SSL (Secure Sockets Layer).

## Client Confidentiality Types

The connector supports the Teradata Client Confidentiality Types corresponding to the different SSL modes and fallback scenarios that are available.

## Deterministic Behavior for Prefer and Allow SSL Modes

When the connector is configured to use SSL in Prefer or Allow mode, the connector supports Teradata deterministic behavior for selecting which port (HTTPS or TDMST) to use during the connection.

During the connection process, the connector discovers which port the database actively listens to, and connects as follows:

- If only the HTTPS port is enabled in the database and the connector's SSL Mode is set to Allow or Prefer, the connector connects to the data store using the HTTPS port.
- If only the TDMST port is enabled in the database and the connector's SSL Mode is set to Allow or Prefer, the connector connects to the data store using the TDMST port.
- If both ports are enabled in the database and the connector's SSL Mode is set to Prefer, the connector connects to the data store using the HTTPS port. If this is unsuccessful, the connector returns an error.
- If both ports are enabled in the database and the connector's SSL Mode is set to Allow, the connector connects to the data store using the TDMST port. If this is unsuccessful, the connector returns an error.

This deterministic behavior is dependent on the TLS setting of your Teradata Gateway. For more information, see the Teradata documentation.

# Data Encryption

The Simba Teradata ODBC Connector supports encryption for any data that is passed between the connector and the database. You can configure the Enable Data Encryption option (the `UseDataEncryption` property) to specify whether the connector encrypts all communication with the database or authentication information only.

# Teradata Wallet

Teradata Wallet is a software package that secures Teradata Database credentials on client machines. It maps your password or authentication parameters to a reference string, which you can then use instead of your actual password or parameters during authentication. Providing reference strings instead of your password or authentication parameters lets you obscure those values.

> **Note:**
> For security reasons, the connector does not allow password values or authentication parameters to be saved in DSNs. You can only save Teradata Wallet reference strings that are mapped to the password or authentication parameters.

Teradata Wallet is installed and configured separately from the connector. To download the software package, go to http://downloads.teradata.com and click the Teradata Wallet link for the platform that you are using. For information about configuring Teradata Wallet, see "Introducting Teradata Wallet" on the Teradata Developer Exchange: http://developer.teradata.com/tools/articles/introducing-teradata-wallet.

After Teradata Wallet is set up, you can connect to the database using the reference strings that are mapped to your password or authentication parameters. When specifying your connection information through the Simba Teradata ODBC Connector DSN Setup dialog box, you can enter your Teradata Wallet reference string directly in the Password Wallet String field. Otherwise, to pass in a reference string to the connector, you must set the `EnableWallet` property to `1`, and then use the following syntax in place of a password value or set of authentication parameters, where *[ReferenceString]* is your reference string:

$tdwallet(*[ReferenceString]*)

For example, the following is a connection string that authenticates the connection using a reference string:

Driver=Simba Teradata ODBC Driver;DBCName=192.168.222.160;
UID=jsmith;WalletString=$tdwallet(jsmith_wallet_string);EnableWallet=1;

# Redrive Support

Redrive is a Teradata Database feature that enables applications to reconnect to the database and resume any interrupted activity after a network error, database failure, or database restart occurs. When connected to a database that has Redrive enabled, the Simba Teradata ODBC Connector can use this feature to reconnect to the database and resume queries after an interruption.

**Note:**
For information about configuring Redrive in the database, see "Redrive Protection for Queries" in *Teradata Database Administration*: https://docs.teradata.com/reader/B7Lgdw6r3719WUyiCSJcgw/tGnoclQ5P79MZYUu52NDdw .

To configure the connector to use Redrive, set the following properties in your DSN or connection string:

- Enable Redrive
- Reconnect Count
- Reconnect Interval

When you connect to the database with Redrive enabled, you have the option of overriding these connection-level settings to enable or disable this feature on the statement level. To do this, set the SQL_ATTR_REDRIVE(13014) statement attribute to one of the following values:

- SQL_REDRIVE_OFF(0) to disable Redrive for the current statement.
- SQL_REDRIVE_ON(1) to enable Redrive for the current statement.

  **Note:**
  You cannot enable Redrive on the statement level if the feature is disabled on the connection level.

# FastExport Support

The FastExport protocol is a Teradata Database feature that improves the performance of SELECT statements that meet certain criteria, enabling you to quickly retrieve large amounts of data. The exact difference in performance varies depending on the application and database configuration, and may be affected by the structure of the query. For example, FastExport works best for queries that do not include any GROUP BY or ORDER BY clauses.

**Note:**
Because the FastExport protocol might return result sets in a different order compared to when the standard protocol is used, in some cases you must use an ORDER BY clause to enforce a specific result set order. For example, without an ORDER BY clause, a query that contains an ordered analytic function might not produce the expected ordered result set.

When connected to a database that supports FastExport, the Simba Teradata ODBC Connector can use the protocol to improve the performance of SELECT queries that meet the following criteria:

- The table or view being queried does not include any data types that the FastExport protocol does not support. For example, BLOB and CLOB data are not supported.
- The query is being run as a prepared statement, using SQLPrepare and SQLExecute. SQLExecDirect queries are not supported.

- If the query is written as multiple semicolon-separated statements, the statements must all be SELECT statements.

Additionally, if you are running more than one prepared statement at the same time, the statements are subject to certain limitations. For more information, see "Restrictions and Limitations" in the *Teradata FastExport Reference*: https://docs.teradata.com/reader/sF_SAf7J~h6dWCL64EJJXQ/qXxSGLsRO0rvHb2qifvxUw.

To configure the connector to use the FastExport protocol, set the following properties in your DSN or connection string:

- Type
- Sessions

# LOB Retrieval Modes

Some Teradata Database instances contain Large Object (LOB) data types, such as BLOB (Binary Large Object) and CLOB (Character Large Object). The Simba Teradata ODBC Connector supports two ways of retrieving LOBs: Deferred Mode and Smart LOB (SLOB) Mode. You can optimize connector performance by configuring the appropriate retrieval mode.

- In Deferred Mode, the connector sends an additional query to retrieve each LOB.
- In SLOB Mode, the connector retrieves LOBs without sending any additional queries, but may need to cache some LOBs in memory. By default, the connector uses SLOB Mode.

To optimize connector performance, use Deferred Mode when retrieving large LOBs that you do not want to cache into memory, and use SLOB Mode when you need to retrieve many small LOBs and want to avoid sending a large number of queries. For example, SLOB Mode can improve connector performance when retrieving geospatial data.

**Important:**
If SLOB Mode is not configured properly, it can decrease connector performance instead of improving it.

# SLOB Mode Usage Guidelines

SLOB Mode is applicable only when certain size restrictions are met:

- The LOB to be retrieved must be smaller than the size specified by the Max Single LOB Bytes (or `MaxSingleLOBBytes`) setting. The connector falls back to using Deferred Mode when retrieving LOBs that exceed this size. By default, the connector uses SLOB Mode for LOB data that is smaller than 4000 bytes.
- If the total amount of LOB data being retrieved from a row exceeds the size specified by the Max Total LOB Bytes Per Row (or `MaxTotalLOBBytesPerRow`) setting, then, after using SLOB Mode to retrieve LOBs up to this size limit, the connector uses Deferred Mode to retrieve the remaining LOBs from that row. By default, the connector can use SLOB Mode to retrieve up to 65536 bytes of LOB data from a row.

When using SLOB Mode, be aware of the following:

- Do not enable the Use Sequential Retrieval Only option (or the `UseSequentialRetrievalOnly` property) if there is any possibility that you might retrieve LOBs from columns in a non-sequential order. For instance, do not enable this option and then execute a query that retrieves LOBs from the third column in a table, then from the first column, and then from the fifth column. If you enable this option and then retrieve LOBs non-sequentially, the connector discards the LOBs that are returned through SLOB Mode and must then retrieve them all again using Deferred Mode.

- When the Use Sequential Retrieval Only option (or the `UseSequentialRetrievalOnly` property) is disabled, the connector caches the other LOBs that it reads while looking for the one to be retrieved. Caching large amounts of data in memory can decrease performance. To prevent this problem, set the size limits so that the connector does not apply SLOB mode to large LOBs. LOB values that do not meet the requirements for SLOB Mode are retrieved using Deferred Mode instead, and therefore do not get cached.

# Controlling the Scope of SLOB Mode Settings

You can configure the settings for SLOB Mode on the connection level or on the statement level. Because the optimal settings vary depending on the size of the specific LOBs that you are retrieving, it may be useful to adjust the settings for each statement as you work with your data.

To configure settings for SLOB Mode on the connection level, specify the relevant connector options in a DSN or connection string. These settings apply to all queries and operations that are executed within the connection. For detailed information about the connector options related to SLOB Mode, see the following:

- Max Single LOB Bytes
- Max Total LOB Bytes Per Row
- Use Sequential Retrieval Only

You can override connection-level settings by using statement attributes. To configure settings for SLOB Mode on the statement level, set the following statement attributes:

- **SQL_ATTR_MAX_SINGLE_LOB_BYTES(13011)**: Use this attribute to specify the maximum size of the LOBs (in bytes) that the connector can retrieve using SLOB Mode. LOBs that exceed this size are retrieved using Deferred Mode instead. This attribute corresponds to the Max Single LOB Bytes (or `MaxSingleLOBBytes`) connector option.

- **SQL_ATTR_MAX_LOB_BYTES_PER_ROW(13012)**: Use this attribute to specify the maximum size of LOB data per row (in bytes) that the connector can retrieve using SLOB Mode. If the total amount of LOB data contained in a row exceeds this size, then the connector retrieves the LOBs from that row using Deferred Mode instead. This attribute corresponds to the Max Total LOB Bytes Per Row (or `MaxTotalLOBBytesPerRow`) connector option.

- **SQL_ATTR_USE_SEQUENTIAL_RETRIEVAL_ONLY(13013)**: Use this attribute to indicate whether you are retrieving LOB data from columns in sequential order. This attribute corresponds to the Use Sequential Retrieval Only (or `UseSequentialRetrievalOnly`) connector option.

# Scalar Function Support

The Simba Teradata ODBC Connector includes full support for all of the scalar functions that are supported by the Teradata Database instance that you connect to. The version of the Teradata Database instance determines which specific scalar functions you can call.

For a list of the scalar functions that are supported by your version of Teradata Database, see the *SQL Functions, Operators, Expressions, and Predicates* book from the Teradata Database documentation set.

When calling a scalar function, it is recommended that you place the function inside an ODBC escape sequence, as this prompts the connector to check if the scalar function is valid before attempting to call it. For example:

```
SELECT {fn MOD(x, y) }
```

For more information about calling scalar functions, see "Scalar Functions" in the *ODBC Driver for Teradata User Guide*.

# Special Query Syntax

The Simba Teradata ODBC Connector includes support for SET TRANSFORM GROUP FOR TYPE statements when connected to a Teradata Database instance that also supports this syntax. This DDL statement enables you to specify the transform group to use on Teradata complex data types (CDTs) that support multiple transform groups on the session level.

Typically, to specify a transform that you want to use for one of these CDTs, you would have to create a user account with the transform settings defined. The SET TRANSFORM GROUP FOR TYPE statement enables you to use the appropriate transform without having to create an additional user account. You can execute this statement multiple times to change transform groups during the same session, if needed.

For detailed information about how to write and execute the SET TRANSFORM GROUP FOR TYPE statement, see "SET TRANSFORM GROUP FOR TYPE Statement" in the *ODBC Driver for Teradata User Guide*.

**Important:**
The SET TRANSFORM GROUP FOR TYPE statement must be executed before the preparation of the main query or after the execution of the main query. If you execute this statement during any other stage of the main query, the connector returns the following error message:

> Error occurred as a SET TRANSFORM GROUP FOR TYPE statement was executed between PREPARE and EXECUTE.

# Connector Configuration Properties

Connector Configuration Options lists the configuration options available in the Simba Teradata 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 Teradata ODBC Driver DSN Setup
- Connector Options
- Advanced Options
- Logging Options

When using a connection string, 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 Teradata ODBC Connector, or via the key name when using a connection string or configuring a connection from a non-Windows machine:

- Account String
- ALL_PROXY
- ALL_PROXY_PASSWORD
- ALL_PROXY_USER
- CODE_APPEND_FILE
- Data Source DNS Entries
- Date Time Format
- Default Database
- Disable Parsing
- Disable Token Cache
- Enable Client Side UDF Upload
- Enable Custom Catalog Mode For 2.x
- Name or IP Address
- No HELP DATABASE
- OIDC_CLIENTID
- OIDC_LOGIN_HINT
- OIDC_PROMPT
- This option specifies the OpenID Connect (OIDC) scope to use for Device Code flow. The default scope can be specified in the database's TdgssUserConfigFile.xml file, using the IdPConfig element's Scope attribute.
- OIDC_TOKEN_CACHE
- Parameter Wallet String
- Password Wallet String

Applications

- Enable Data Encryption
- Enable Event Tracing for Windows
- Enable Extended Statement Info
- Enable Read Ahead
- Enable Redrive
- EXTERNALBROWSER Tab Timeout
- HTTPS Port Number
- HTTP_PROXY
- HTTP_PROXY_PASSWORD
- HTTP_PROXY_USER
- HTTPS_PROXY
- HTTPS_PROXY_PASSWORD
- HTTPS_PROXY_USER
- Ignore Search Patterns
- This option specifies the JSON Web Signature (JWS) algorithm to sign the JWT Bearer Token for client authentication.
- JWS_CERT
- This option specifies the file name of the PEM or JWK file containing the private key.
- Log Level
- Log Path
- Login Timeout
- Max File Size
- Procedure With Print Stmt
- Procedure With SPL Source
- Reconnect Count
- Reconnect Interval
- Retry System Calls (EINTR)
- Return Empty String In CREATE_ PARAMS Column For SQL_ TIMESTAMP
- Return Generated Keys
- Return Max CHAR/VARCHAR Length As 32k
- Return Output Parameters As Result Set
- Session Character Set
- Session Mode
- Sessions
- SSL CA Path
- SSL CA File Name
- SSL Mode
- SSL Protocol
- TDMST Port Number
- Translation DLL Name
- Translation Option
- Type
- UDF Upload Path
- UPT Mode
- Use Column Names

# Account String

The account string to use when logging in to the database.

| Key Name | Default Value | Required |
|---|---|---|
| `AccountStr`<br>OR<br>`Account` | The account string that is associated with the specified user name. | No |

# ALL_PROXY

This option specifies the host name or IP address of the proxy server, supports both HTTP and HTTPS proxy.

| Key Name | Default Value | Required |
|---|---|---|
| `ALL_PROXY` | None | No |

# ALL_PROXY_PASSWORD

This option specifies the proxy server password for the ALL_PROXY server.

| Key Name | Default Value | Required |
|---|---|---|
| `ALL_PROXY_PASSWORD` | None | No |

# ALL_PROXY_USER

This option specifies the proxy server username for the ALL_PROXY server.

| Key Name | Default Value | Required |
|---|---|---|
| ALL_PROXY_USER | None | No |

# CODE_APPEND_FILE

This option specifies how you want the verification URL and code to display.

Set this option to one of the following values:

- -out: The connector prints the verification URL and code to C++ std::cout.
- -err: The connector prints the verification URL and code to C++ std::cerr.
- [*filename*]: The connector appends the verification URL and code to the specified file if the file already exists or creates a new file if the specified file does not exist.

| Key Name | Default Value | Required |
|---|---|---|
| CODE_APPEND_FILE | -out | No |

# Data Source DNS Entries

This option specifies how the connector determines which DNS entry to connect to.

- If this option is not set, the connector resolves DNS entries dynamically.
- If this option is set to 0, the connector uses DNS lookup.
- If this option is set to a non-zero value, then the connector uses that number of DNS entries in a round-robin fashion.

| Key Name | Default Value | Required |
|---|---|---|
| DataSourceDNSEntries | None | No |

# Date Time Format

This option specifies the format that the connector uses for DATE values when communicating with the database.

- AAA: The connector uses ANSI format for DATE values.
- IAA: The connector uses Integer format for DATE values.

| Key Name | Default Value | Required |
|---|---|---|
| DateTimeFormat | AAA | No |

# Default Database

The name of the database to access by default.

If this option is not set, then the connector uses the default database assigned to the specified user name.

If a table owner is not specified, then all catalog functions are associated with the default database.

| Key Name | Default Value | Required |
|---|---|---|
| `DefaultDatabase` | The default database that is associated with the specified user name. | No |

# Disable Parsing

This option specifies whether the connector parses SQL statements or passes the statements through to the database without making any modifications.

- Enabled (`1`): SQL statements are passed through to the Teradata Database without any modifications.
- Disabled (`0`): SQL statements are parsed by the connector.

| Key Name | Default Value | Required |
|---|---|---|
| `NoScan` | Clear (`0`) | No |

# Disable Token Cache

This option specifies whether the connector reuses the credentials and log in multiple times for short-lived connections without refreshing tokens to authenticate the connection.

- Enabled (`No`): The connector uses the tokens from the token memory cache.
- Disabled (`Yes`): The connector does not use token memory cache and it will behave the same as connectors prior to `20.00.00.019`.

| Key Name | Default Value | Required |
|---|---|---|
| `DisableTokenCache` | No | No |

# Enable Client Side UDF Upload

This option specifies whether the connector supports the creating and updating of User-Defined Functions (UDFs) on the Teradata Database server based on UDFs that are saved in files on the client machine.

- Enabled (`1`): The connector supports UDF file uploads. You can create or update UDFs on the server by calling CREATE FUNCTION or REPLACE FUNCTION, with an EXTERNAL NAME

clause specifying the files where the UDFs are defined.

> **Important:**
> If this option is enabled, then you must also set the UDF Upload Path (or `UDFUploadPath`) option. For more information, see UDF Upload Path.

- Disabled (`0`): The connector does not support UDF file uploads.

For information about the supported statement syntax, see "CREATE FUNCTION and REPLACE FUNCTION (External Form)" in *SQL Data Definition Language Detailed Topics* from Teradata: https://info.teradata.com/HTMLPubs/DB_TTU_16_00/index.html#page/SQL_Reference%2FB035-1184-160K%2Fbcb1472241301533.html%23.

| Key Name | Default Value | Required |
|---|---|---|
| `EnableUDFUpload` | Clear (`0`) | No |

# Enable Custom Catalog Mode For 2.x Applications

This option provides backwards compatibility for ODBC 2.x applications that use noncompliant search patterns.

Earlier versions of the connector allowed users to create search patterns other than the `%` search pattern stated in the ODBC Programmer's Reference specification. On noncompliant systems, if a NULL value is passed to the SQLTables API for the SchemaName argument, the result is a search for tables by userid, DBC, and default database schema names, rather than the `%` search pattern.

- Enabled (`1`): The connector allows searches by userid, DBC, and default database schema names.
- Disabled (`0`): The connector uses the `%` search pattern.

| Key Name | Default Value | Required |
|---|---|---|
| `Use2xAppCustomCatalogMode` | Clear (`0`) | No |

# Enable Data Encryption

This setting determines if the connection encrypts authentication data exclusively or all database communications. The database's ClientAttributeEx can now have its data encryption value updated from the connection string key UseDataEncryption.

- Enabled (`1`): The connector encrypts all data that is passed between the connector and the database.
- Disabled (`0`): The connector encrypts authentication information only.

| Key Name | Default Value | Required |
|---|---|---|
| UseDataEncryption | Clear (0) | No |

# Enable Event Tracing for Windows

This option specifies whether the connector logs information to the Event Viewer of the Teradata server.

- Enabled: Error events are logged to the Event Viewer.
- Disabled: Error events are not logged to the Event Viewer.

> **Note:**
> - This option is available only in the Windows connector.
> - This option is a connector-wide configuration option, so its setting applies to all connections that use the Simba Teradata ODBC Connector, and it cannot be set as a connection string property.

| Key Name | Default Value | Required |
|---|---|---|
| N/A | Clear | No |

# Enable Extended Statement Info

This option specifies whether extended statement information is used when it is available from the database (Teradata Database versions V2R6.2 and later).

- Enabled (1): Extended statement information is requested and used, and the ODBC API function SQLDescribeParam is supported.
- Disabled (0): Extended statement information is not used, and the ODBC API function SQLDescribeParam is not supported.

| Key Name | Default Value | Required |
|---|---|---|
| EnableExtendedStmtInfo | Selected (1) | No |

# Enable Read Ahead

This option specifies whether to request the next response message while the current message is being processed.

- Enabled (1): The connector requests the next response message while the current message is being processed.
- Disabled (0): The connector does not request the next response message until the current message has been processed.

| Key Name | Default Value | Required |
|---|---|---|
| EnableReadAhead | Selected (1) | No |

# Enable Redrive

This option specifies whether the connector uses the Redrive feature. Redrive enables the connector to attempt to reconnect to the database and resume activities after an interruption has occurred, such as from a network error, database failure, or database restart.

**Note:**

- In order for the connector to use Redrive, the feature must also be enabled on the database that you are connecting to. For more information, see "Redrive Protection for Queries" in *Teradata Database Administration*: https://docs.teradata.com/reader/B7Lgdw6r3719WUyiCSJcgw/tGnoclQ5P79MZYUu52NDdw.
- When you connect with Redrive enabled, you can enable or disable the feature on the statement level by using the SQL_ATTR_REDRIVE(13009) statement attribute. For more information, see Redrive Support.

Set this option to one of the following values:

- `Yes`: The connector attempts to reconnect and resume activities after the connection is interrupted.
- `No`: The connector does not attempt to reconnect to the database after the connection is interrupted.
- `Default`: The connector determines whether or not to use Redrive based on the configuration of the database. For example, if Redrive is enabled on the database, then the connector uses Redrive.

For information about configuring the behavior of the connector during reconnection attempts, see Reconnect Count and Reconnect Interval.

| Key Name | Default Value | Required |
|---|---|---|
| EnableRedrive | Default | No |

# EXTERNALBROWSER Tab Timeout

This option specifies the amount of time, in seconds, to wait before the connector closes the browser tab after Federated Authentication is completed. If enabled, the tab used to log on remains open, but the second and subsequent tabs are automatically closed.

**Note:**
Not all browsers support automatic closing of browser tabs.

Set this option to one of the following values:

- `-1`: The connector does not close browser tabs automatically.
- `0`: The connector closes the browser tab immediately.
- A number greater or equal to `1`: The connector waits the specified seconds to close the browser tab.

| Key Name | Default Value | Required |
|---|---|---|
| `BrowserTabTimeout` | `5` | No |

# HTTP_PROXY

This option specifies the host name or IP address of the HTTP proxy server.

| Key Name | Default Value | Required |
|---|---|---|
| `HTTP_PROXY` | None | No |

# HTTP_PROXY_PASSWORD

This option specifies the proxy server password for the HTTP_PROXY server.

| Key Name | Default Value | Required |
|---|---|---|
| `HTTP_PROXY_PASSWORD` | None | No |

# HTTP_PROXY_USER

This option specifies the Proxy server username for the HTTP_PROXY server.

| Key Name | Default Value | Required |
|---|---|---|
| `HTTP_PROXY_USER` | None | No |

# HTTPS_PROXY

This option specifies the host name or IP address of the HTTPS proxy server.

| Key Name | Default Value | Required |
|---|---|---|
| `HTTPS_PROXY` | None | No |

# HTTPS_PROXY_PASSWORD

This option specifies the proxy server password for the HTTPS_PROXY server.

| Key Name | Default Value | Required |
|---|---|---|
| HTTPS_PROXY_PASSWORD | None | No |

# HTTPS Port Number

The number of the TCP port that the Teradata uses to access the database when using SSL.

| Key Name | Default Value | Required |
|---|---|---|
| HTTPS_PORT | 443 | No |

# HTTPS_PROXY_USER

This option specifies the Proxy server username for the HTTPS_PROXY server.

| Key Name | Default Value | Required |
|---|---|---|
| HTTPS_PROXY_USER | None | No |

# Ignore Search Patterns

This option specifies whether the underscore (_) and percent sign (%) characters are parsed as normal characters or as search wildcards.

- Enabled (1): The underscore (_) and percent sign (%) characters are parsed as normal characters.
- Disabled (0): The underscore (_) and percent sign (%) characters are parsed as ODBC search wildcards.

| Key Name | Default Value | Required |
|---|---|---|
| IgnoreODBCSearchPattern | Clear (0) | No |

# JWS_ALGORITHM

This option specifies the JSON Web Signature (JWS) algorithm to sign the JWT Bearer Token for client authentication.

Set this option to one of the following values:

- RS256: specifies RSASSA-PKCS1-v1_5 using SHA-256.
- RS384 specifies RSASSA-PKCS1-v1_5 using SHA-384.
- RS512 specifies RSASSA-PKCS1-v1_5 using SHA-512.

| Key Name | Default Value | Required |
|---|---|---|
| JWS_ALGORITHM | RS256 | No |

# JWS_CERT

This option specifies the file name of the X.509 certificate PEM file that contains the public key corresponding to the private key from JWS_PRIVATE_KEY.

When this parameter is specified, the x5t header thumbprint is added to the JWT Bearer Token for the Identity Provider to select the public key for JWT signature verification. Some Identity Providers, such as Microsoft Entra ID, require this.

When this parameter is omitted, the x5t header thumbprint is not added to the JWT Bearer Token. Some Identity Providers do not require the x5t header thumbprint.

| Key Name | Default Value | Required |
|---|---|---|
| JWS_CERT | None | No |

# JWS_PRIVATE_KEY

This option specifies the file name of the PEM or JWK file containing the private key.

Based on the Identity Provider (IDP) used, two types of private key files are supported:

- `.jwk`: If this file contains a kid (key identifier) parameter, the kid header is added to the JWT Bearer Token for the Identity Provider to select the public key for JWT signature verification.
- `.pem`: This file must contain the BEGIN/END PRIVATE KEY header and trailer.

| Key Name | Default Value | Required |
|---|---|---|
| JWS_PRIVATE_KEY | None | Yes, if using Client Credentials authentication. |

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

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, where *[UserName]* and *[ProcessID]* are the user name and process ID associated with the connection that is being logged, and *[Number]* is a number that identifies each log file:

- A *[UserName]_[ProcessID]*_simbateradataodbcconnector.log file that logs connector activity that is not specific to a connection.
- A *[UserName]_[ProcessID]*_simbateradataodbcconnector_connection_*[Number]*.log file for each connection made to the database.

| 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:**
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.teradataodbc.ini file.

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

# Login Timeout

The number of seconds that the connector waits for a response when logging in to the database.

| Key Name | Default Value | Required |
|---|---|---|
| LoginTimeout | 20 | No |

# 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:**
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.teradataodbc.ini` file.

| 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:**
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.teradataodbc.ini` file.

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

# Max Single LOB Bytes

The maximum size of the LOBs (in bytes) that the connector can retrieve using Smart LOB (SLOB) Mode. LOBs that exceed this size are retrieved using Deferred Mode instead.

If this option is set to `0`, SLOB Mode is disabled, and the connector retrieves all LOB data using Deferred Mode. For more information, see LOB Retrieval Modes.

**Note:**
As an alternative to using this option, you can specify this setting on the statement level rather than the connection level by using the SQL_ATTR_MAX_SINGLE_LOB_BYTES statement attribute.

| Key Name | Default Value | Required |
|---|---|---|
| `MaxSingleLOBBytes` | `4000` | No |

# Max Total LOB Bytes Per Row

The maximum size of LOB data per row (in bytes) that the connector can retrieve using Smart LOB (SLOB) Mode. If the total amount of LOB data contained in a row exceeds this size, then the connector retrieves the LOBs from that row using Deferred Mode instead.

If this option is set to `0`, SLOB Mode is disabled, and the connector retrieves all LOB data using Deferred Mode. For more information, see LOB Retrieval Modes.

**Note:**
As an alternative to using this option, you can specify this setting on the statement level rather than the connection level by using the SQL_ATTR_MAX_LOB_BYTES_PER_ROW statement attribute.

| Key Name | Default Value | Required |
|---|---|---|
| `MaxTotalLOBBytesPerRow` | `65536` | No |

# Maximum Response Buffer

The maximum size of the response buffer for SQL requests, in kilobytes.

When you are connected to a database instance that is running Teradata Database 16.00 or later, the maximum value is 7361536. For connections that use earlier versions of Teradata Database, the maximum value is 1048576.

| Key Name | Default Value | Required |
|---|---|---|
| `MaxRespSize` | `524288` | No |

# Mechanism

The mechanism that the connector uses to authenticate the connection to the database.

Select one of the following settings, or set the key to the name of the authentication mechanism:

- **KRB5**: The connector uses the Kerberos protocol. The application provides the user name and password.
- **LDAP**: The connector uses the LDAP protocol. The application provides the user name and password.
- **TD2**: The connector uses the Teradata 2 mechanism, which requires you to provide a Teradata Database user name and password. For information about the options that you use to specify your credentials, see the following:
  - Username
  - Password Wallet String
  - Password

- **TDNEGO**: The connector uses the mechanism that is selected automatically through Teradata Negotiating, which can include single sign-on.
- **JWT**: The connector uses a JSON web token (JWT) to authenticate the connection.
- **EXERNALBROWSER**: The connector uses CloudSSO to authenticate the connection. The user's identity is obtained through Keycloak or PingFederate login using an external browser. The user is logged on without providing a username and password.

**Note:**
If this option is not set, then the connector uses the authentication mechanism specified in the `tdgssconfigure.xml` file in the TeraGSS program. This is typically TD2.

| Key Name | Default Value | Required |
|---|---|---|
| `MechanismName` | None | No |

# Name or IP Address

The fully qualified domain name or IP address of the Teradata Database instance.

| Key Name | Default Value | Required |
|---|---|---|
| `DBCName` | None | Yes |

# No HELP DATABASE

This option specifies whether the Help Database is used.

- Enabled (`1`): SQLTables uses a SELECT statement when no wildcard characters are used in SQLTables.
- Disabled (`0`): The connector uses the HELP DATABASE command.

**Note:**
If this option is enabled, then SQLTables uses either `dbc.tables` or `dbc.tablesX`, depending on whether X Views are enabled. For more information about X Views, see Use X Views.

| Key Name | Default Value | Required |
|---|---|---|
| `DonotUseHelpDatabase` | Clear (`0`) | No |

# OIDC_CLIENTID

This option specifies the OpenID Connect (OIDC) Client ID. When the OIDC_CLIENTID connection parameter is not set, the default Client ID comes from the database's `TdgssUserConfigFile.xml` file.

| Key Name | Default Value | Required |
|---|---|---|
| OIDC_CLIENTID | Client ID from TdgssUserConfigFile.xml | No |

# OIDC_LOGIN_HINT

This option specifies which username to log in as.

| Key Name | Default Value | Required |
|---|---|---|
| OIDC_LOGIN_HINT | None | No |

# OIDC_PROMPT

This option tells the Identity Provider to provide appropriate prompts.

| Key Name | Default Value | Required |
|---|---|---|
| OIDC_PROMPT | None | No |

# OIDC_SCOPE

This option specifies the OpenID Connect (OIDC) scope to use for Device Code flow. The default scope can be specified in the database's TdgssUserConfigFile.xml file, using the IdPConfig element's Scope attribute.

| Key Name | Default Value | Required |
|---|---|---|
| OIDC_SCOPE | openid | No |

# OIDC_TOKEN_CACHE

This option enables token caching in memory.

| Key Name | Default Value | Required |
|---|---|---|
| OIDC_TOKEN_CACHE | None | No |

# Parameter Wallet String

| Key Name | Default Value | Required |
|---|---|---|
| AuthenticationParameter<br>OR<br>MechanismKey | None | Yes, if authenticating using a JSON web token (JWT). |

Additional parameters that might be required for authentication. For example, if you are authenticating your connection to the database using a JSON web token (JWT), then you must specify a token parameter.

Depending on whether you are connecting using a DSN or a connection string, you need to use different methods to specify your parameters:

- When setting this option in a DSN, you must specify a Teradata Wallet reference string that is mapped to your parameters. You cannot specify the parameters directly in the DSN.
- When setting this option in a connection string, you may choose to specify your parameters directly in the connection string.

Additionally, when specifying a reference string in a connection string or in the `odbc.ini` configuration file, you must enclose the reference string inside the `$tdwallet()` syntax. If you are specifying the reference string in the Simba Teradata ODBC Connector DSN Setup dialog box instead, you can choose to specify the reference string value without that syntax.

**Note:**

- The Teradata Wallet utility must be installed and configured before you can connect to the database using a reference string. For more information, see Teradata Wallet.
- When specifying a parameter directly in a connection string, if the parameter contains any of the following special characters, you must enclose the parameter in braces ({}): `* @ [] {} , = ! () ? ;`

For example, depending on whether you are connecting using a connection string or a DSN, you would provide a JWT using one of the following ways:

- When configuring a DSN through the Simba Teradata ODBC Connector DSN Setup dialog box, in the Parameter Wallet String field, type the reference string. You can choose whether or not to include the `$tdwallet()` syntax.
- When configuring a DSN using the `odbc.ini` file, set the following connection property, where *[ReferenceString]* is the reference string:

  AuthenticationParameter=$tdwallet(*[ReferenceString]*)
- When writing a connection string, you can set the `AuthenticationParameter` property to the reference string, using the following syntax where *[ReferenceString]* is the reference string:

  AuthenticationParameter=$tdwallet(*[ReferenceString]*)

  Or, you can set the `AuthenticationParameter` property to the token parameter, using the following syntax where *[JWT_Token]* is the JWT value:

  AuthenticationParameter={token=*[JWT_Token]*}

# Password Wallet String

The Teradata Wallet reference string that is mapped to your Teradata Database password.

When setting this option in a connection string or in the `odbc.ini` configuration file, you must enclose the reference string inside the `$tdwallet()` syntax.

For example, where *[ReferenceString]* is the reference string:

WalletString=$tdwallet(*[ReferenceString*])

When setting this option in the Simba Teradata ODBC Connector DSN Setup dialog box, you may choose to specify the reference string without that syntax.

> **Note:**
> The Teradata Wallet utility must be installed and configured before you can connect using a reference string. For more information, see Teradata Wallet.

| Key Name | Default Value | Required |
|---|---|---|
| `WalletString` | None | No |

# Procedure With Print Stmt

This option specifies whether to enable the print option for stored procedures.

- `P`: The SPL PRINT statements specified in the stored procedure body are saved in the compiled stored procedure.
- `N`: The SPL PRINT statements are not saved. If the Procedure With SPL Source option (the `SplOption` property) is enabled, then the connector preserves the SPL PRINT statements in the SPL source text.

| Key Name | Default Value | Required |
|---|---|---|
| `PrintOption` | `N` | No |

# Procedure With SPL Source

This option specifies whether to use stored procedure language (SPL) when creating stored procedures.

- `Y`: SPL is enabled, and the source text must be stored in Teradata Database.
- `N`: SPL is disabled, and the source text is not stored in the server.

| Key Name | Default Value | Required |
|---|---|---|
| `SplOption` | `Y` | No |

# PROXY_BYPASS_HOSTS

This option specifies a comma separated list of host name, domain, and IP address.

| Key Name | Default Value | Required |
|---|---|---|
| PROXY_BYPASS_HOSTS | None | No |

# Reconnect Count

The maximum number of times that the connector tries to reconnect to the database after the connection has been interrupted.

The largest supported value for this property is 99.

| Key Name | Default Value | Required |
|---|---|---|
| ReconnectCount | 20 | No |

# Reconnect Interval

The number of seconds that the connector waits between reconnection attempts, when trying to reconnect to the database after the connection has been interrupted.

The largest supported value for this property is 300.

| Key Name | Default Value | Required |
|---|---|---|
| ReconnectInterval | 30 | No |

# Retry System Calls (EINTR)

This option specifies whether the connector retries the socket system calls or returns a SQL_ERROR when an EINTR error occurs.

- Enabled (1): The connector retries the socket system calls.
- Disabled (0): The connector returns a SQL_ERROR, and the ODBC application becomes responsible for recovering from the interrupted socket system calls.

| Key Name | Default Value | Required |
|---|---|---|
| RetryOnEINTR | Selected (1) | No |

# Return Empty String In CREATE_PARAMS Column For SQL_TIMESTAMP

This option specifies whether the connector returns an empty string or the given value for the CREATE_PARAMS column when you call SQLGetTypeInfo for SQL_TIMESTAMP data.

- Enabled (1): The connector returns an empty string, and prohibits Microsoft Access from using any TIMESTAMP precision values when creating tables.

- Disabled (`0`): The connector returns the given value.

> **Note:**
> This option is applicable only for Windows and macOS.

| Key Name | Default Value | Required |
|---|---|---|
| `UseEmptyCreateParamsColumnForTimestamp` | Clear (`0`) | No |

# Return Generated Keys

This option determines the result from requests that insert data into identity columns. These requests can optionally return a result set containing identity column values, also known as auto-generated keys, for the inserted rows.

- `C`: The connector retrieves the identity columns only.
- `R`: The connector retrieves the entire row.
- `N`: The connector does not retrieve auto-generated keys.

| Key Name | Default Value | Required |
|---|---|---|
| `ReturnGeneratedKeys` | `N` | No |

# Return Max CHAR/VARCHAR Length As 32k

This option specifies whether the connector returns a hard-coded value for the COLUMN_SIZE column when you call SQLGetTypeInfo for SQL_CHAR and SQL_VARCHAR data. Enabling this option prevents the returned column size from causing numeric overflows in Microsoft Access.

- Enabled (`1`): The connector returns a hard-coded value for the maximum size of SQL_CHAR and SQL_VARCHAR columns.
- Disabled (`0`): The connector returns the actual maximum size of the column. In some cases, Microsoft Access might experience numeric overflow when processing the column size returned by the connector.

Depending on the Session Character Set (or `CharacterSet`) setting, the hard-coded value is 32000 or 64000. For more information, see Session Character Set.

> **Note:**
> This option is applicable only for Windows and macOS.

| Key Name | Default Value | Required |
|---|---|---|
| `Use32kMaxCharColumnSize` | Clear (`0`) | No |

# Return Output Parameters As Result Set

This option specifies whether the connector returns output parameters as a result set.

- Enabled (`1`): The connector returns output parameters as a result set.
- Disabled (`0`): The connector does not return output parameters as a result set.

| Key Name | Default Value | Required |
|---|---|---|
| `OutputAsResultSet` | Clear (`0`) | No |

# Session Character Set

The character set to use for the session. This value can be a user-defined character set, or one of the following pre-defined character sets:

- `ASCII`
- `UTF8`
- `UTF16`
- `LATIN1252_0A`
- `LATIN9_0A`
- `LATIN1_0A`
- `Shift-JIS` (Windows, DOS compatible, `KANJISJIS_0S`)
- `EUC` (Unix compatible, `KANJIEC_0U`)
- `IBM Mainframe` (`KANJIEBCDIC5035_0I`)
- `KANJI932_1S0`
- `BIG5` (`TCHBIG5_1R0`)
- `GB` (`SCHGB2312_1T0`)
- `SCHINESE936_6R0`
- `TCHINESE950_8R0`
- `NetworkKorean` (`HANGULKSC5601_2R4`)
- `HANGUL949_7R0`
- `ARABIC1256_6A0`
- `CYRILLIC1251_2A0`

- HEBREW1255_5A0
- LATIN1250_1A0
- LATIN1254_7A0
- LATIN1258_8A0
- THAI874_4A0

**Note:**
The specified character set must be installed on Teradata Database.

| Key Name | Default Value | Required |
|---|---|---|
| CharacterSet | ASCII | No |

# Session Mode

This option specifies the session mode that the connector uses during sessions on the database.

- ANSI: The connector uses ANSI mode.
- System Default: The connector uses the default session mode of the system that you are using the connector on.
- Teradata: The connector uses Teradata mode.

| Key Name | Default Value | Required |
|---|---|---|
| SessionMode | System Default | No |

# Sessions

The number of FastExport data connections that the connector opens, to enable performance improvements for SELECT queries that meet the criteria of the FastExport protocol.

The number of AMPs (Access Module Processors) that are available for your database determines the maximum number of FastExport data connections that can be opened. If you set this property to a number that is greater than the number of AMPs, the connector only opens a number of connections equal to the number of AMPs.

**Note:**
We recommend that you do not set this property. When this property is not set, the number of FastExport connections is determined automatically based on the database settings.

For more information about FastExport, see FastExport Support.

| Key Name | Default Value | Required |
|---|---|---|
| Sessions | None | No |

# SSL CA Path

The full path of the directory containing the root certificates for trusted CAs, for verifying the server when using SSL.

**Note:**
This setting is applicable only when SSLMode is set to Verify-CA or Verify-Full.

| Key Name | Default Value | Required |
|---|---|---|
| SSLCAPath | None | No |

# SSL CA File Name

The full path and file name of the .pem file containing trusted Root and Intermediate CA certificates, for verifying the server when using SSL.

**Note:**
This setting is applicable only when SSLMode is set to Verify-CA or Verify-Full.

| Key Name | Default Value | Required |
|---|---|---|
| SSLCA | The default for the operating environment. | No |

# SSL Mode

The level of security (SSL/TLS) that the connector uses for the connection to the data store.

- Allow (Allow): The connector connects to the data store using the TDMST port if it is enabled; if not, the connector uses the HTTPS port.

  If both ports are enabled, the connector connects to the data store using the TDMST port. If this is unsuccessful, the connector returns an error.

- Disable (Disable): The connector only connects to the data store using the TDMST port.

- Prefer (Prefer): The connector connects to the data store using the HTTPS port if it is enabled; if not, the connector uses the TDMST port.

  If both ports are enabled, the connector connects to the data store using the HTTPS port. If this is unsuccessful, the connector returns an error.

- Require (Require): The connector only connects to the data store using the HTTPS port.

- Verify-CA (`Verify-CA`): The connector only connects to the data store using the HTTPS port. In addition, the connector verifies the server CA certificate against the configured CA certificates.
- Verify-Full (`Verify-Full`): The connector only connects to the data store using the HTTPS port. In addition, the connector verifies the server CA certificate against the configured CA certificates, and performs additional host name identity verification.

**Note:**
If this property is set to `Allow` or `Prefer`, and the Teradata Gateway is set to Enable SSL connections, the connector only attempts to connect using the TDMST port. If this is unsuccessful, the connector returns an error. For more information, see "Deterministic Behavior of Prefer/Allow SSLMode" in the Teradata TLS Websocket documentation.

| Key Name | Default Value | Required |
|---|---|---|
| `SSLMode` | Prefer (`Prefer`) | No |

# SSL Protocol

This property specifies the TLS protocol version used.

Currently, only TLS v1.2 is supported.

| Key Name | Default Value | Required |
|---|---|---|
| `SSLProtocol` | TLSv1.2 (`TLSv1.2`) | No |

# TDMST Port Number

The number of the port used to access Teradata Database.

**Important:**
Do not change this value unless instructed to do so by Technical Support.

| Key Name | Default Value | Required |
|---|---|---|
| `TdmstPortNumber` | `1025` | No |

# Translation DLL Name

The full path to the `.dll` file that contains functions for translating all the data that is transferred between the Teradata server and the connector.

This `.dll` file is used for translation if local character sets are not supported by Teradata Database or the connector.

| Key Name | Default Value | Required |
|---|---|---|
| TranslationDllName | None | No |

# Translation Option

The options used by the Translation DLL file (see Translation DLL Name). The required options may vary depending on the Translation DLL file being used.

| Key Name | Default Value | Required |
|---|---|---|
| TranslationOption | None | No |

# Type

This option specifies whether the connector uses the FastExport protocol to improve the performance of SELECT queries that meet certain criteria.

- Default: The connector does not use FastExport for any queries, and only runs queries using the standard protocol.
- FastExport: When connected to a database that supports FastExport, the connector uses it to run queries that meet the FastExport criteria. For all other queries, the connector falls back to using the standard protocol.

For information about the requirements for using FastExport, see FastExport Support.

| Key Name | Default Value | Required |
|---|---|---|
| Type | Default | No |

# UDF Upload Path

**Note:**

- This option is applicable only when the Enable Client Side UDF Upload (or EnableUDFUpload) option is enabled. For more information, see Enable Client Side UDF Upload.
- The connector does not accept relative paths.

The full path to a directory on the client machine that contains UDF files. The connector automatically prepends this path to the file names that you specify in the EXTERNAL NAME clauses for CREATE FUNCTION or REPLACE FUNCTION calls. This enables you to write EXTERNAL NAME clauses that only specify file names.

If your UDF files are stored in various different directories and you want to specify the full path and name of the files when writing EXTERNAL NAME clauses, then set this option to an empty string.

**Important:**
We recommend setting this option to a specific directory path instead of an empty string. Empty strings might not be supported in future releases.

Setting this option to an empty string can present a security risk. Doing so enables the connector to upload UDF files from any directory, so it is possible for a third party to modify the path and cause an unexpected UDF file to be uploaded to the database.

| Key Name | Default Value | Required |
|---|---|---|
| `UDFUploadPath` | `Please enter the UDF folder path` | Yes, if Enable Client Side UDF Upload (or `EnableUDFUpload`) is enabled. |

# UPT Mode

This option specifies whether the connector supports Unicode Pass Through (UPT) for Pass Through Characters (PTCs). For more information about UPT, see "Unicode Pass Through" in the Teradata Database documentation: http://info.teradata.com/htmlpubs/DB_TTU_16_00/index.html#page/General_Reference/B035-1098-160K/ifk1472240714022.html.

- Notset (`NOTSET`): The connector does not do anything to change UPT support.
- UPTON (`UPTON`): The connector sends a query to the database to enable UPT support. When UPT support is enabled, the connector allows PTCs to be passed through to the database.
- UPTOFF (`UPTOFF`): The connector sends a query to the database to disable UPT support. When UPT support is disabled, the connector does not allow PTCs to be passed through to the database.

| Key Name | Default Value | Required |
|---|---|---|
| `UPTMode` | Notset (`NOTSET`) | No |

# Use Column Names

This option specifies whether column names or column titles are returned.

- Enabled (`1`): The connector returns column names.
- Disabled (`0`): The connector returns column titles if they are defined. Otherwise, the connector returns column names.

**Note:**
Column titles for SQLColumns are shown in the LABEL column.

| Key Name | Default Value | Required |
|---|---|---|
| DontUseTitles | Selected (1) | No |

# Use DATE Data for TIMESTAMP Parameters

This option specifies whether the connector sends DATE data for parameters that are bound as SQL_TIMESTAMP or SQL_C_TIMESTAMP.

**Important:**
This option should only be enabled for applications that use Microsoft Access Jet databases, as it can result in truncating SQL_C_TIMESTAMP data.

- Enabled (1): The connector sends DATE data for SQL_TIMESTAMP and SQL_C_TIMESTAMP parameters.
- Disabled (0): The connector sends standard data for these parameters.

| Key Name | Default Value | Required |
|---|---|---|
| UseDateDataForTimeStampParams | Clear (0) | No |

# Use Integrated Security

This option specifies whether the connector authenticates the connection using Single Sign-On (SSO) or Conventional Sign-On (CSO).

- Enabled (1): The connector uses SSO and authenticates the connection by using Teradata Database credentials that are derived from the user information on your client machine.
- Disabled (0): The connector uses CSO and requires you to provide your Teradata Database credentials.

| Key Name | Default Value | Required |
|---|---|---|
| UseIntegratedSecurity | Clear (0) | No |

# Use NULL For Catalog Name

This option specifies whether the connector sets any Catalog Name parameters to NULL.

- Enabled (1): Catalog Name parameters are set to NULL for all Catalog API functions, even if the application passes a value.
- Disabled (0): Catalog Name parameter values are passed in. In this case the connector returns an error, because Teradata Database does not support catalogs.

| Key Name | Default Value | Required |
|---|---|---|
| TABLEQUALIFIER | Clear (0) | No |

# Use Regional Settings for Decimal Symbol

This option specifies whether the connector uses the regional settings for decimal symbols, or uses a period (.) regardless of the regional settings.

- Enabled (1): The connector uses the regional settings for decimal symbols.
- Disabled (0): The connector uses a period (.) for decimal symbols regardless of the regional settings.

**Note:**
This option is applicable only for Windows and macOS.

| Key Name | Default Value | Required |
|---|---|---|
| UseRegionalSettings | Selected (1) | No |

# Use Sequential Retrieval Only

This option indicates to the connector whether you are retrieving LOB data from columns in sequential order or non-sequential order. When working in Smart LOB (SLOB) Mode, the connector reads and caches LOB data differently depending on this setting. For more information about SLOB Mode, see LOB Retrieval Modes.

- Enabled (1): When working in SLOB Mode, the connector does not cache the other LOBs that it reads while looking for the one to be retrieved. Because the connector can retrieve LOBs in a single pass if they are queried sequentially, the connector does not need to cache them.
- Disabled (0): When working in SLOB Mode, the connector caches the other LOBs that it reads while looking for the one to be retrieved. This caching allows the connector to successfully retrieve SLOBs in any order.

**Important:**

- Do not enable this option if there is any possibility that you might retrieve LOBs from columns in a non-sequential order. For instance, do not enable this option and then execute a query that retrieves LOBs from the third column in a table, then from the first column, and then from the fifth column. If you enable this option and then retrieve LOBs non-sequentially, the connector discards the LOBs that are returned through SLOB Mode and must then retrieve them all again using Deferred Mode.
- As an alternative to using this option, you can specify this setting on the statement level rather than the connection level by using the SQL_ATTR_USE_SEQUENTIAL_RETRIEVAL_ONLY statement attribute.

| Key Name | Default Value | Required |
|---|---|---|
| `UseSequentialRetrievalOnly` | Clear (`0`) | No |

# Use TCP_NODELAY

This option specifies whether TCP immediately sends small packets or waits to gather packets into a single, larger packet.

- Enabled (`1`): TCP immediately sends small packets. This option can avoid transmission delays but might increase network traffic.
- Disabled (`0`): TCP gathers small packets into a single larger packet. This option can reduce network traffic but might cause transmission delays.

| Key Name | Default Value | Required |
|---|---|---|
| `TcpNoDelay` | Selected (`1`) | No |

# Use X Views

This option specifies whether to use X views. X views restrict access to the data so that the connector can only access objects that the specified user owns or controls.

- Enabled (`1`): The connector uses the following views:
  - SQLTables() and SQLProcedures() use `dbc.tablesVX` and `dbc.databasesVX`
  - SQLColumns() and SQLProcedureColumns() use `dbc.columnsVX`
  - SqlStatistics() uses `dbc.tablesizeVX`
- Disabled (`0`): The connector uses the following views:
  - SQLTables() and SQLProcedures() use `dbc.tablesV` and `dbc.databasesV`
  - SQLColumns() and SQLProcedureColumns() use `dbc.columnsV`
  - SqlStatistics() uses `dbc.tablesizeV`

| Key Name | Default Value | Required |
|---|---|---|
| `UseXViews` | Clear (`0`) | No |

# Username

Your user name for authenticating the connection to Teradata Database through the specified authentication mechanism. For example, if you set the **Mechanism** option to **TD2** (set the `MechanismName` key to `TD2`), then you must provide your Teradata Database user name.

| Key Name | Default Value | Required |
|---|---|---|
| `UID`<br><br>OR | None | Yes, if the authentication mechanism is TD2. |

| Key Name | Default Value | Required |
|---|---|---|
| Username | | |

# Configuration Options Having Only Key Names

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

- DataSourceName / DSN
- DriverLocale
- Driver
- EnableWallet
- IANAAppCodePage
- IntervalPeriodTypesToString
- Password

## DataSourceName / DSN

The name of the DSN that you want to use to connect to Teradata Database.

**Note:**
This property is used in connection strings only. It cannot be set in the `odbc.ini` file.

| Key Name | Default Value | Required |
|---|---|---|
| DataSourceName<br>OR<br>DSN | None | No |

## Driver

In Windows, the name of the installed connector for (`Simba Teradata 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 ODBC Driver` when installed in Windows, or the absolute path of the connector | Yes |

| Key Name | Default Value | Required |
|---|---|---|
| | shared object file when installed on a non-Windows machine. | |

# DriverLocale

The locale to use for error messages.

Set this property to one of the following values:

- `en-US`: The connector returns error messages in English.
- `ja-JP`: The connector returns error messages in Japanese.

| Key Name | Default Value | Required |
|---|---|---|
| `DriverLocale` | `en-US` | No |

# EnableWallet

This option specifies whether the connector authenticates the connection using a Teradata Wallet reference string instead of a password.

- `1`: The connector uses a Teradata Wallet reference string.
- `0`: The connector uses a password.

| Key Name | Default Value | Required |
|---|---|---|
| `EnableWallet` | `0` | Yes, if using a Teradata Wallet reference string instead of a password. |

# IANAAppCodePage

The ODBC application code page that the connector uses when converting characters between ANSI and Unicode.

For a list of supported values, see "ODBC Application Code Page Values" in Teradata's *ODBC Driver for Teradata User Guide*.

**Note:**

- This property is applicable only for macOS and Unix.
- This setting takes precedence over the `CharacterSet` setting. For information about the `CharacterSet` setting, see Session Character Set.

| Key Name | Default Value | Required |
|---|---|---|
| IANAAppCodePage | None | No |

# IntervalPeriodTypesToString

This option specifies whether the connector returns interval, time zone, and XML data types from the Teradata Database as strings, or as SQL types that map more closely to the Teradata data type.

This option applies to the following Teradata data types:

- INTERVAL DAY
- INTERVAL DAY TO HOUR
- INTERVAL DAY TO MINUTE
- INTERVAL DAY TO SECOND
- INTERVAL HOUR
- INTERVAL HOUR TO MINUTE
- INTERVAL HOUR TO SECOND
- INTERVAL MINUTE
- INTERVAL MINUTE TO SECOND
- INTERVAL MONTH
- INTERVAL SECOND
- INTERVAL YEAR
- INTERVAL YEAR TO MONTH
- TIME WITH TIME ZONE
- TIMESTAMP WITH TIME ZONE
- XML

Set this option to one of the following values:

- `1`: The connector returns these Teradata data types as SQL_WLONGVARCHAR data.
- `0`: The connector returns these Teradata data types as SQL_INTERVAL types, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, or custom SQL types. For information about the exact data type mappings, see Data Types.

**Note:**
Some applications, such as .NET applications, do not support custom SQL data types.

| Key Name | Default Value | Required |
|---|---|---|
| `IntervalPeriodTypesToString` | `0` | No |

# Password

The password that you use to access the database.

**Note:**

- This property is applicable in connection strings only. For security reasons, the connector does not allow password values to be saved in DSNs.
- To provide your password in a DSN, you must use the Teradata Wallet utility to map your password to a reference string, and then set the Password Wallet String (or `WalletString`) property to the reference string. For more information, see Password Wallet String and Teradata Wallet.

| Key Name | Default Value | Required |
|---|---|---|
| `Password` | None | Yes, if the authentication mechanism is TD2, and you are using a connection string that does not specify a reference string for the database password. |

# Third-Party Trademarks

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.

Teradata is a trademark or registered trademark of Teradata Corporation or its subsidiaries in Canada, the United States and/or other countries.

All other trademarks are trademarks of their respective owners.