insightsoftware

# Simba PostgreSQL ODBC Data Connector

Installation and Configuration Guide

Version 1.6

November 2025

# Contents

# Copyright



## Contact Us

www.insightsoftware.com

# About This Guide

## Purpose

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

## Audience

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

## Knowledge Prerequisites

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

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

## Document Conventions

*Italics* is used when referring to book and document titles.

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

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

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

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

# About the Simba PostgreSQL ODBC Connector

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

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

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

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

# Platform and Data source version support

The Simba PostgreSQL ODBC Connector supports Windows, macOS , and Linux operating systems. For details on the specific versions of operating systems supported as well as the versions of the data source supported, please refer to the connector's release notes.

# Windows Connector

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

# Windows System Requirements

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

- Administrator rights on your machine.
- 100 MB of available disk space

Before the connector can be used, the Visual C++ Redistributable for Visual Studio 2022 with the same bitness as the connector must also be installed. If you obtained the connector from the Simba website, then your installation of the connector automatically includes this dependency. Otherwise, you must install the redistributable manually. You can download the installation packages for the redistributable at https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170.

# 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 PostgreSQL <Version Number> 32-bit.msi` for 32-bit applications
- `Simba PostgreSQL <Version Number> 64-bit.msi` for 64-bit applications

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

To install the Simba PostgreSQL ODBC Connector in Windows:

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

# Creating a Data Source Name in Windows

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

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

To create a Data Source Name in Windows:

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

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

2. In the ODBC Data Source Administrator, click the **Drivers** tab, and then scroll down as needed to confirm that the Simba PostgreSQL ODBC Driver appears in the alphabetical list of ODBC connectors that are installed on your system.
3. Choose one:

- To create a DSN that only the user currently logged into Windows can use, click the **User DSN** tab.
- Or, to create a DSN that all users who log into Windows can use, click the **System DSN** tab.

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

4. Click **Add**.
5. In the Create New Data Source dialog box, select **Simba PostgreSQL ODBC Driver** and then click **Finish**. The Simba PostgreSQL ODBC Driver DSN Setup dialog box opens.
6. In the **Data Source Name** field, type a name for your DSN.
7. In the **Server** field, type a comma-delimited list of endpoint servers you want to connect to.
8. In the **Port** field, type the number of the TCP port that the server uses to listen for client connections.

   **Note:** The default port used by PostgreSQL is 5432.

9. In the **Database** field, type the name of the database that you want to access.
10. In the **Authentication** area, do one of the following:

    - To authenticate the connection using standard user name and password authentication, do the following:

- From the **Auth Type** drop-down list, select **Standard**.
- In the **User** field, type your user name for accessing the PostgreSQL database.
- In the **Password** field, type the password corresponding to the user name you typed.
- To encrypt your credentials, select one of the following:

a. If the credentials are used only by the current Windows user, select **Current User Only**.

b. Or, if the credentials are used by all users on the current Windows machine, select **All Users Of This Machine**.

- Or, to authenticate the connection using the Kerberos protocol, do the following:
  - From the **Auth Type** drop-down list, select **Kerberos**.
  - In the **User** field, type your user name for accessing the PostgreSQL database.
  - In the **Service Name** field, type the Kerberos service principal name of the PostgreSQL server.

11. To configure client-server verification over SSL, click **SSL Options**. For more information, see Configuring SSL Verification in Windows.
12. To configure advanced connector options, click **Additional Options**. For more information, see Configuring Additional Options in Windows.
13. To configure logging behavior for the connector, click **Logging Options**. For more information, see Configuring Logging Options in Windows.
14. To configure how the connector returns and displays data, click **Data Type Options**. For more information, see Configuring Data Type Options in Windows.
15. To test the connection, click **Test**. Review the results as needed, and then click **OK**.

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

16. To save your settings and close the Simba PostgreSQL ODBC Driver DSN Setup dialog box, click **OK**.
17. To close the ODBC Data Source Administrator, click **OK**.

# Configuring SSL Verification in Windows

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

To configure SSL verification in Windows:

1. To access the SSL options for a DSN, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **SSL Options**.
2. In the **Authentication Mode** list, select the appropriate SSL mode.

   > **Note:**
   > For information about SSL support in PostgreSQL, see "SSL Support" in the PostgreSQL Documentation: http://www.postgresql.org/docs/9.4/static/libpq-ssl.html.

3. To specify the minimum version of SSL to use, from the **Minimum TLS** drop-down list, select the minimum version of SSL.
4. To use the System Trust Store for SSL certificates, select the **Use System Trust Store** check box.
5. If you selected **Use System Trust Store**, choose one of the following options:
   - To check the validity of the certificate's trust chain, select the **Check Certificate Revocation** check box.
   - Or, to accept self-signed certificates, select the **Allow Self-signed Server Certificate** check box.
6. To specify an SSL certificate, select the **Enable Custom SSL CA Root Certificate** check box, and then, in the **Path** field, specify the full path to the certificate file.
7. To specify the client SSL certificate, in the **Client Cert** field, type the full path of the file containing the client SSL certificate.
8. To specify the client SSL key path, in the **Client Key** field, type the full path of the file containing the secret key used for the client certificate.
9. To save your settings and close the dialog box, click **OK**.
10. To save your settings and close the Simba PostgreSQL ODBC Driver DSN Setup dialog box, click **OK**.

# Configuring Data Type Options in Windows

You can configure data type options to modify how the connector displays or returns some data types.

To configure data type options in Windows:

1. To access data type options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Data Type Options**.
2. To enable the connector to return data as Unicode character types, select the **Use Unicode** check box.

> **Note:**
> When the **Use Unicode** check box is selected, the connector does the following:
>
> - Returns SQL_WCHAR instead of SQL_CHAR.
> - Returns SQL_WVARCHAR instead of SQL_VARCHAR.
> - Returns SQL_WLONGVARCHAR instead of SQL_LONGVARCHAR.

3. To configure the connector to return Boolean columns as SQL_VARCHAR instead of SQL_BIT, select the **Show Boolean Column As String** check box.
4. To configure the connector to return Text columns as SQL_LONGVARCHAR instead of SQL_VARCHAR, select the **Text as LongVarChar** check box.
5. To configure the connector to return Bytea columns as SQL_LONGVARBINARY instead of SQL_VARBINARY, select the **Bytea As LongVarBinary** check box.
6. To configure the connector to return Money columns with non-decimal characters removed, select the **Money As Decimal** check box.
7. In the **Max Varchar** field, type the maximum data length for VarChar columns.
8. In the **Max LongVarChar** field, type the maximum data length for LongVarChar columns.
9. In the **Max Bytea** field, type the maximum data length for Bytea columns.
10. To save your settings and close the Data Type Configuration dialog box, click **OK**.

# Configuring Additional Options in Windows

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

To configure additional options in Windows:

1. To access advanced options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Additional Options**.
2. Specify how the connector processes queries by doing one of the following:
   - To return query results one row at a time, select **Single Row Mode**.
   - To return a specific number of rows at a time, select **Use Declare/Fetch** and then, in the **Cache Size** field, type the number of rows.
   - To enable the connector to have more than one query, separated by a semicolon ( `;` ), in a single SQLExecDirect call, select **Use Multiple Statements**.

- To return the entire query result, select **Retrieve Entire Result Into Memory**.

> **Note:**
> Use **Single Row Mode** if you plan to query large results and you do not want to retrieve the entire result into memory. Disabling **Single Row Mode** increases performance, but can result in out-of-memory errors.

3. To configure the connector to return SQL_ERROR immediately for any other queries that is executed if there is already an active query in execution under the same connection, select the **Enforce Single Statement** check box.
4. To configure the connector to recognize table type information from the data source, select the **Enable Table Types** check box. For more information, see Enable Table Types.
5. To configure the connector to enable read-only mode, select the **Enable Read Only** check box. For more information, see Enable Read Only.
6. To connect to PostgreSQL through a proxy server, select the **Enable Proxy For PostgreSQL Connection** check box and then do the following:
   a. In the **Proxy Server** field, type the host name or IP address of the proxy server.
   b. In the **Proxy Port** field, type the number of the TCP port that the proxy server uses to listen for client connections.
   c. If the proxy server requires authentication, then do the following:
   i. In the **Proxy Username** field, type your user name for accessing the proxy server.
   ii. In the **Proxy Password** field, type the password corresponding to the user name.
7. To save your settings and close the Additional Configuration dialog box, click **OK**.
8. To save your settings and close the Simba PostgreSQL ODBC Driver DSN Setup dialog box, click **OK**.

# Configuring TCP Keepalives in Windows

By default, the Simba PostgreSQL ODBC Connector is configured to use TCP keepalives to prevent connections from timing out. Settings such as how frequently the connector sends TCP keepalive packets are based on the operating system defaults. You can configure the TCP keepalive settings or disable the feature by modifying the appropriate values in the Windows Registry.

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

To configure TCP keepalives in Windows:

1. On the Start screen, type **regedit**, and then click the **regedit** search result.
2. Select the appropriate registry key for the bitness of your connector:
   - If you are using the 32-bit connector on a 64-bit machine, then select the following registry key, where *[YourDSN]* is the DSN for which you want to configure keepalives:

     **HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBC.INI\ *[YourDSN]***

   - Otherwise, select the following registry key, where *[YourDSN]* is the DSN for which you want to configure keepalives:

     **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI*[YourDSN]***

3. To specify the interval of inactivity before the connector sends a TCP keepalive packet, configure the **KeepAliveIdle** value by doing the following:
   a. If the **KeepAliveIdle** value does not already exist, create it. Select **Edit > New > String Value**, type **KeepAliveIdle** as the name of the value, and then press **Enter**.
   b. Select the **KeepAliveIdle** value, and then Select **Edit > Modify**.
   c. In the Edit String dialog box, in the **Value Data** field, type the number of seconds of inactivity before the connector sends a TCP keepalive packet.

      **Note:** To use the system default, in the **Value Data** field, type **0**.

   d. Click **OK**.
4. To specify the number of TCP keepalive packets that can be lost before the connection is considered broken, configure the KeepAliveCount value. To do this, follow the procedure above, but type **KeepAliveCount** for the value name, and in the **Value Data** field, type the number of keepalive packets that can be lost.

   **Note:** To use the system default, in the **Value Data** field, type **0**.

5. To specify the interval of time between each retransmission of a keepalive packet, configure the KeepAliveInterval value. To do this, follow the procedure above, but type **KeepAliveInterval** for the value name, and in the **Value Data** field, type the number of seconds to wait between each retransmission.

   **Note:** To use the system default, in the **Value Data** field, type **0**.

6. Close the Registry Editor.

To disable TCP keepalives:

1. On the Start screen, type **regedit**, and then click the **regedit** search result.
2. Select the appropriate registry key for the bitness of your connector:
   - If you are using the 32-bit connector on a 64-bit machine, then select the following registry key, where `[YourDSN]` is the DSN for which you want to configure keepalives:

     **HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBC.INI*[YourDSN]***

   - Otherwise, select the following registry key, where `[YourDSN]` is the DSN for which you want to configure keepalives:

     **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI*[YourDSN]***
3. If the **KeepAlive** value does not already exist, create it. Select **Edit > New > String Value**, then type **KeepAlive** as the name of the value, and then press **Enter**.
4. Select the **KeepAlive** value, and then click **Edit > Modify**.
5. In the Edit String dialog box, in the **Value Data** field, type **0**.
6. Click **OK**.
7. Close the Registry Editor.

> **Note:**
> To enable TCP keepalives after disabling them, set `KeepAlive` to `1`.

# Configuring Logging Options in Windows

To help troubleshoot issues, you can enable logging. In addition to functionality provided in the Simba Simba PostgreSQL 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 PostgreSQL ODBC Connector, so make sure to disable the feature after you are done using it. To configure logging for the current connection, see Configuring Logging for the Current Connection.

To enable connector-wide logging in Windows:

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

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

3. In the **Log Path** field, specify the full path to the folder where you want to save log files.
4. Click **OK**.
5. Restart your ODBC application to make sure that the new settings take effect.

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

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

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

To disable connector logging in Windows:

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

# Configuring Logging for the Current Connection

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

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

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

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

**To add logging configurations to a DSN in Windows:**

1. On the Start screen, type **regedit**, and then click the **regedit** search result.
2. Navigate to the appropriate registry key for the bitness of your connector and your machine:
   - 32-bit System DSNs: **HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\ODBC\ODBC.INI*[DSN Name]***
   - 64-bit System DSNs: **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI*[DSN Name]***
   - 32-bit and 64-bit User DSNs: **HKEY_CURRENT_USER\SOFTWARE\ODBC\ODBC.INI*[DSN Name]***
3. For each configuration option that you want to configure for the current connection, create a value by doing the following:
   a. If the key name value does not already exist, create it. Right-click the *[DSN Name]* and then select **New > String Value**, type the key name of the configuration option, and then press **Enter**.
   b. Right-click the key name and then click **Modify**.

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

# Verifying the Connector Version Number in Windows

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

To verify the connector version number in Windows:

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

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

2. Click the **Drivers** tab and then find the Simba PostgreSQL ODBC Connector in the list of ODBC Connectors that are installed on your system. The version number is displayed in the **Version** column.

# macOS Connector

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

# macOS System Requirements

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

- 105.5MB of available disk space
- One of the following ODBC driver managers installed:
  - iODBC 3.52.15 or later
  - unixODBC 2.3.9 or later

# 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 PostgreSQL ODBC Connector is available for macOS as a `.dmg` file named `Simba PostgreSQL 1.6.dmg`. The connector only supports 64-bit client applications.

To install the Simba PostgreSQL ODBC Connector in macOS:

1. Double-click **Simba PostgreSQL <Version Number> .dmg** to mount the disk image.
2. Double-click **Simba PostgreSQL <Version Number> .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/postgresqlodbc` 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/postgresqlodbc/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 PostgreSQL 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.postgresqlodbc
```

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

# Linux Connector

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

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

# Linux System Requirements

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

- 150MB of available disk space
- If dynamically linked libraries are installed:
  - libstdc++ version 6.0.21 or later
  - GCC 5.5 or later
- One of the following ODBC driver managers installed:
  - iODBC 3.52.9 or later
  - unixODBC 2.2.14 or later
- glibc 2.17 or later

For information about libstdc++, including installation instructions, see "Chapter 2. Setup" in The GNU C++ Library Manual: https://gcc.gnu.org/onlinedocs/libstdc++/manual/setup.html.

- The `krb5-libs` library that matches the bitness of the connector must be installed.

> **Note:** If the package manager in your Linux distribution cannot resolve the dependency automatically when installing the connector, then download and manually install the package.

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

# Installing the Connector Using the RPM File

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

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

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

The placeholders in the file names are defined as follows:

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

**Note:** For Linux ARM, only the 64-bit version of the connector is supported.

**To install the Simba PostgreSQL ODBC Connector using the RPM File:**

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

   - Or, if you are using Linux ARM Server, navigate to the `Linux_ARM` folder containing the RPM and run the following command:

     ```
     Linux ARM/zypper install [RPMFileName]
     ```

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

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

**To install the Simba PostgreSQL ODBC Connector using the RPM File (Linux ARM):**

1. Log in as the root user.
2. Follow the instructions in the installer to complete the installation process. The Simba PostgreSQL ODBC Connector files are installed in the `/opt/simba/postgresqlodbc` directory.
3. If you received a license file via email, then copy the license file into the `/opt/simba/postgresqlodbc/64` folder. You must have root privileges when changing the contents of this folder.

# Installing the Connector Using the Tarball Package

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

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

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

**To install the connector using the tarball package:**

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

   ```
   tar -zxvf [TarballName]
   ```

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

   The Simba PostgreSQL ODBC Connector files are installed in the `/opt/simba/postgresqlodbc` 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 in Linux

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

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

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

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

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

1. Navigate to the `/lib` subfolder in your connector installation directory. By default, the path to this directory is: `/opt/simba/postgresqlodbc/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 PostgreSQL ODBC Connector, do the following:

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

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

# Specifying ODBC Driver Managers in Non-Windows Machines

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

## macOS

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

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

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

## Linux

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

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

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

# 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.postgresqlodbc.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 SIMBAPOSTGRESQLODBCINI to the full path and file name of the `simba.postgresqlodbc.ini` file.

> **Note:** If you accquired the connector from a vendor other than Simba, you need to replace SIMBA with the name of your vendor.

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 SIMBAPOSTGRESQLODBCINI to the full path and file name of the `simba.postgresqlodbc.ini` file.

> **Note:** If you accquired the connector from a vendor other than Simba, you need to replace SIMBA with the name of your vendor.

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

**For unixODBC:**

```
export ODBCINI=/usr/local/odbc/odbc.ini
export ODBCSYSINI=/usr/local/odbc
export SIMBAPOSTGRESQLODBCINI=/etc/simba.postgresqlodbc.ini
```

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

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

# Configuring ODBC Connections in Non-Windows Machine

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

- Creating a Data Source Name on a Non-Windows Machine
- Configuring SSL Verification on a Non-Windows Machine
- Configuring Query Processing Modes on a Non-Windows Machine
- Configuring a Proxy Connection on a Non-Windows Machine
- Configuring TCP Keepalives 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.

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

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

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

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

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

   For example, on a macOS machine:

   [ODBC Data Sources]

   Sample DSN=Simba PostgreSQL ODBC Driver

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

   [ODBC Data Sources]

Sample DSN=Simba PostgreSQL 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/postgresqlodbc/lib/libpostgresqlodbc_sbu.dylib

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

      Driver=/opt/simba/postgresqlodbc/lib/32/libpostgresqlodbc_sb32.so

   b. Set the `Server` property to a comma-delimited list of endpoint servers you want to connect to, and then set the `Port` property to the number of the TCP port that these servers use to listen for client connections.

      For example:

      Server=testserver.abcabcabcabc.com,testserver.cbacbacba.com,

      Port=5439

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

      For example:

      Database=TestDB

   d. To configure authentication, do one of the following:

      - To use standard authentication, do the following:
        - Set the `UseKerberos` property to `0`.
        - Set the `UID` property to your user name for accessing the PostgreSQL server.
        - Set the `PWD` property to the password corresponding to the user name that you provided in the previous step.

      For example:

      UseKerberos=0

      UID=simba

      PWD=simba123

      - Or, to use Kerberos authentication, do the following:
        - Set the `UseKerberos` property to `1`.
        - Set the `UID` property to your user name for accessing the PostgreSQL server.
        - Set the `KerberosServiceName` property to the Kerberos service principal name of the PostgreSQL server.

For example:

UseKerberos=1

UID=simba

KerberosServiceName=simba_postgresql

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

f. Optionally, modify how the connector runs queries and retrieves results into memory. For more information, see Configuring Query Processing Modes on a Non-Windows Machine.

g. Optionally, configure the connector to connect through a proxy server. For more information, see Configuring a Proxy Connection on a Non-Windows Machine.

h. Optionally, modify the TCP keepalive settings that the connector uses to prevent connections from timing out. For more information, see Configuring TCP Keepalives on a Non-Windows Machine.

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

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

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

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

[ODBC Data Sources]

Sample DSN=Simba PostgreSQL ODBC Driver

[Sample DSN]

Driver=/Library/simba/postgresqlodbc/lib/libpostgresqlodbc_sbu.dylib

Server=192.168.222.160

Port=5432

Database=TestDB

UseKerberos=0

UID=simba

PWD=simba123

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

[ODBC Data Sources]

Sample DSN=Simba PostgreSQL ODBC Driver 32-bit

[Sample DSN]

Driver=/opt/simba/postgresqlodbc/lib/32/libpostgresqlodbc_sb32.so

Server=192.168.222.160

Port=5432

Database=TestDB

UseKerberos=0

UID=simba

PWD=simba123

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

# Configuring a DSN-less Connection in a Non-Windows Machine

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

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

To define a connector on a non-Windows machine:

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

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

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

   For example:

   [ODBC Drivers]

   Simba PostgreSQL ODBC Connector=Installed

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

   For example, on a macOS machine:

   Driver=/Library/simba/postgresqlodbc/lib/libpostgresqlodbc_sbu.dylib

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

   Driver=/opt/simba/postgresqlodbc/lib/32/libpostgresqlodbc_sb32.so

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

For example:

Description=Simba PostgreSQL ODBC Connector

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

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

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

[ODBC Drivers]

Simba PostgreSQL ODBC Connector=Installed

[Simba PostgreSQL ODBC Connector]

Description=Simba PostgreSQL ODBC Connector

Driver=/Library/simba/postgresqlodbc/lib/libpostgresqlodbc_sbu.dylib

As another example, the following is an `odbcinst.ini` configuration file for both the 32- and 64-bit connectors in Linux:

[ODBC Drivers]

Simba PostgreSQL ODBC Connector 32-bit=Installed

Simba PostgreSQL ODBC Connector 64-bit=Installed

[Simba PostgreSQL ODBC Connector 32-bit]

Description=Simba PostgreSQL ODBC Connector (32-bit)

Driver=/opt/simba/postgresqlodbc/lib/32/libpostgresqlodbc_sb32.so

[Simba PostgreSQL ODBC Connector 64-bit]

Description=Simba PostgreSQL ODBC Connector (64-bit)

Driver=/opt/simba/postgresqlodbc/lib/64/libpostgresqlodbc_sb64.so

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

# Configuring SSL Verification on a Non-Windows Machine

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

SSL, the connector supports identity verification between the client and the server.

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

To configure SSL verification on a non-Windows machine:

1. Set the `SSLMode` property to the appropriate SSL mode.

   **Note:**
   For information about SSL support in PostgreSQL, see "SSL Support" in the PostgreSQL Documentation: http://www.postgresql.org/docs/9.4/static/libpq-ssl.html.

2. To specify an SSL certificate, set the `SSLCertPath` property to the full path and file name of the certificate file.
3. To specify the client SSL certificate, set the `SSLClientCertPath` property to the full path of the file containing the client SSL certificate.
4. To specify the client SSL key path, set the `SSLClientKeyPath` property to the full path of the file containing the secret key used for the client certificate.
5. To specify the minimum version of SSL to use, set the `Min_TLS` property to the minimum version of SSL. Supported options include `1.0` for TLS 1.0, `1.1` for TLS 1.1, and `1.2` for TLS 1.2.

# Configuring Query Processing Modes on a Non-Windows Machine

To optimize connector performance, you can modify how the connector runs queries and retrieves results into memory. For example, you can configure the connector to return entire query results into memory all at once, or one row at a time. Use a query processing mode that prevents queries from consuming too much memory, based on the expected result size of your queries and the specifications of your system.

**Note:**
Use Single Row Mode if you plan to query large results and you do not want to retrieve the entire result into memory. Using the other query processing modes increases performance, but can result in out-of-memory errors.

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.

# Enabling Single Row Mode

You can configure the connector to return query results one row at a time.

To enable Single Row Mode:

1. Set the `SingleRowMode` property to `1`.
2. Make sure that the `UseDeclareFetch` property is set to `0` or not set.

# Enabling Declare/Fetch Mode

You can configure the connector to return a specific number of rows at a time.

To enable Declare/Fetch Mode:

1. Set the `UseDeclareFetch` property to `1`.
2. Set the `Fetch` property to the number of rows that the connector returns at a time.

# Enabling Retrieve Entire Result Mode

You can configure the connector to return entire query results into memory.

To enable Retrieve Entire Result Mode:

- Make sure that the `SingleRowMode`, `UseDeclareFetch`, and `UseMultipleStatements` properties are set to `0` or not set.

# Enabling Multiple Statements Mode

The connector can have more than one query, separated by a semicolon ( `;` ), in a single SQLExecDirect call. The connector returns all the query results into memory.

To enable Multiple Statements Mode:

1. Set the `UseMultipleStatements` property to `1`.
2. Make sure that the `SingleRowMode` and `UseDeclareFetch` properties are set to `0` or not set.

# Enabling Enforce Single Statement Mode

You can configure the connector to return SQL_ERROR immediately for any other queries that is executed if there is already an active query in execution under the same connection.

To enable Enforce Single Statement Mode:

1. Set the `EnforceSingleStatement` property to `1`.
2. Make sure that the `UseMultipleStatements` is set to `0` or not set.

# Configuring a Proxy Connection on a Non-Windows Machine

You can configure the connector to connect to PostgreSQL through a proxy server, so that communications between the connector and your PostgreSQL data source are passed through the proxy server.

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

To configure a proxy connection on a non-Windows machine:

1. Set the `ProxyHost` property to the host name or IP address of the proxy server.
2. Set the `ProxyPort` property the number of the TCP port that the proxy server uses to listen for client connections.
3. If the proxy server requires authentication, then do the following:
   a. Set the `ProxyUid` property to your user name for accessing the proxy server.
   b. Set the `ProxyPwd` property to the password corresponding to the user name.

# Configuring TCP Keepalives on a Non-Windows Machine

By default, the Simba PostgreSQL ODBC Connector is configured to use TCP keepalives to prevent connections from timing out. Settings such as how frequently the connector sends TCP keepalive packets are based on the operating system defaults.

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 TCP keepalives on a non-Windows machine:

1. Set the `KeepAliveIdle` property to the number of seconds of inactivity before the connector sends a TCP keepalive packet.
2. Set the `KeepAliveCount` property to the number of keepalive packets that can be lost before the connection is considered broken.
3. Set the `KeepAliveInterval` property to the number of seconds to wait before each retransmission of a keepalive packet.

> **Note:**
> To use the system default for `KeepAliveIdle`, `KeepAliveCount`, or `KeepAliveInterval`, set the property to `0`.

To disable TCP keepalives:

- Set the `KeepAlive` property to `0`.

> **Note:**
> To enable TCP keepalives after disabling them, remove the `KeepAlive` property or set it to `1`.

# 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.
- The settings for logging apply to every connection that uses the Simba PostgreSQL ODBC Connector, so make sure to disable the feature after you are done using it.

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

To enable logging on a non-Windows machine:

1. Open the `simba.postgresqlodbc.ini` configuration file in a text editor.
2. 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. |

3. Set the `LogPath` key to the full path to the folder where you want to save log files.
4. Optionally, to prefix the log file name with the user name and process ID associated with the connection, set the `UseLogPrefix` property to `1`.
5. Save the `simba.postgresqlodbc.ini` configuration file.
6. Restart your ODBC application to make sure that the new settings take effect.

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

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

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

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

1. Open the `simba.postgresqlodbc.ini` configuration file in a text editor.
2. Set the `LogLevel` key to `0`.
3. Save the `simba.postgresqlodbc.ini` configuration file.
4. Restart your ODBC application to make sure that the new settings take effect.

# Testing the Connection in Non-Windows Machine

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

## Using the iODBC Driver Manager

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

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

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

**To test your connection using the iODBC driver manager:**

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

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

## Using the unixODBC Driver Manager

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

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

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

**To test your connection using the unixODBC driver manager:**

- Run isql or iusql by using the corresponding syntax:

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

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

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

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

# Using a Connection String

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

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

# DSN Connection String Example

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

DSN=*[DataSourceName]*

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

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

# DSN-less Connection String Examples

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

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

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

- *[DatabaseName]* is the database that you want to access.
- *[PortNumber]* is the number of the TCP port that the PostgreSQL server uses to listen for client connections.
- *[PPort]* is the number of the TCP port that the proxy server uses to listen for client connections.
- *[PServer]* is the IP address or host name of the proxy server to which you are connecting.
- *[Server]* is the IP address or host name of the PostgreSQL server to which you are connecting.
- *[ServiceName]* is the Kerberos service principal name of the PostgreSQL server.

- *[YourPassword]* is the password corresponding to your user name.
- *[YourUserName]* is the user name that you use to access the PostgreSQL server.

# Connecting to a PostgreSQL Server Using Standard Authentication

The following is the format of a DSN-less connection string for a basic connection to a PostgreSQL server:

Driver=Simba PostgreSQL ODBC Driver;
Server=*[Server]*;Port=*[PortNumber]*;
Database=*[DatabaseName]*;UseKerberos=0;
UID=*[YourUserName]*;PWD=*[YourPassword]*;

For example:

Driver=Simba PostgreSQL ODBC Driver;Server=192.168.222.160;
Port=5400;Database=TestDB;UseKerberos=0;UID=simba;
PWD=simba;

# Connecting to a PostgreSQL Server Using Kerberos Authentication

The following is the format of a DSN-less connection string for connecting to a PostgreSQL using Kerberos authentication:

Driver=Simba PostgreSQL ODBC Driver;
Server=*[Server]*;Port=*[PortNumber]*;
Database=*[DatabaseName]*;UseKerberos=1;
UID=*[YourUserName]*;KerberosServiceName=*[ServiceName]*;

For example:

Driver=Simba PostgreSQL ODBC Driver;Server=192.168.222.160;
Port=5400;Database=TestDB;UseKerberos=1;
UID=simba;KerberosServiceName=simba_postgresql;

# Connecting to a PostgreSQL Server Through a Proxy Server

The following is the format of a DSN-less connection string for connecting to a PostgreSQL server through a proxy server:

Driver=Simba PostgreSQL ODBC Driver;
Server=*[Server]*;Port=*[PortNumber]*;
Database=*[DatabaseName]*;UseKerberos=0;
UID=*[YourUserName]*;PWD=*[YourPassword]*;
ProxyHost=*[PServer]*;ProxyPort=*[PPort]*;

For example:

Driver=Simba PostgreSQL ODBC Driver;Server=192.168.222.160;
Port=5400;Database=TestDB;UseKerberos=0;UID=simba;
PWD=simba;ProxyHost=192.168.222.160;

ProxyPort=8000;

# Features

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

- Query Processing Modes
- TCP Keepalives
- Data Types
- Security and Authentication

# Query Processing Modes

To support performance tuning, the Simba PostgreSQL ODBC Connector provides different query processing modes that you can configure to modify how the connector runs queries and retrieves results into memory.

The following query processing modes are available:

- **Single Row Mode**: The connector returns query results one row at a time.
- **Declare/Fetch Mode**: The connector returns a user-specified number of rows at a time.
- **Retrieve Entire Result Mode**: The connector returns the entire query result into memory.
- **Multiple Statements Mode:** The connector can have more than one query, separated by a semicolon ( `;` ), in a single SQLExecDirect call. The application calls SQLMoreResults to move to the next result set. When using this mode, the connector returns all the query results into memory.
- **Enforce Single Statement Mode**: The connector allows applications to allocate more than one statement handle and execute queries in each statement handle concurrently per connection. However, the connector allows only one active statement at a time for each connection. When using this mode, the connector returns SQL_ERROR immediately for any other queries that is executed if there is already an active query in execution under the same connection. You can use this mode in conjunction with the Single Row, Declare/Fetch, and Retrieve Entire Result modes. For more information, see Enforce Single Statement.

By default, the connector does not allow more than one active query at a time, and returns the entire query result into memory. When there is an active query in execution, the connector blocks queries in other statement handles from execution until the active query finishes execution and retrieves all the data, or when the application calls SQLCloseCursor or SQLFreeHandle with a HandleType of SQL_Handle_STMT to indicate that the statement handle can be freed.

Use a query processing mode that prevents queries from consuming too much memory, considering the expected result size of your queries and the specifications of your system.

For information about configuring how the connector processes queries, see Configuring Additional Options in Windows if you are using the Windows version of the connector, or see Configuring Query

Processing Modes on a Non-Windows Machine if you are using a non-Windows version of the connector.

# TCP Keepalives

By default, the Simba PostgreSQL ODBC Connector is configured to use TCP keepalives to verify the status of a connection and prevent it from timing out. After you connect to a PostgreSQL server, the connector automatically sends keepalive packets to the server. If the server does not respond, then the connector returns an indication that the connection is broken.

For information about configuring settings for TCP keepalives when using the Windows connector, see Configuring TCP Keepalives in Windows. For information about configuring settings for TCP keepalives when using the Linux or macOS connector, see Configuring TCP Keepalives on a Non-Windows Machine.

# Data Types

The Simba PostgreSQL ODBC Connector supports many common data formats, converting between PostgreSQL data types and SQL data types.

The table below lists the supported data type mappings.

**Note:**
If the Use Unicode option (the `UseUnicode` key) is enabled, then the connector returns SQL_WCHAR instead of SQL_CHAR, and SQL_WVARCHAR instead of SQL_VARCHAR.

| PostgreSQL Type | SQL Type |
|---|---|
| ARRAY | SQL_VARCHAR |
| BIGINT | SQL_BIGINT |
| BIGSERIAL | SQL_BIGINT |
| BIT | SQL_VARCHAR<br>■ If the length of the column is greater than the Max Varchar (`MaxVarchar`) setting, then SQL_LONGVARCHAR is returned instead.<br>■ If the Use Unicode option (the `UseUnicode` key) is enabled, then SQL_WVARCHAR is returned instead.<br>■ If the Use Unicode option (the `UseUnicode` key) is enabled and the column length is greater than the Max Varchar (`MaxVarchar`) setting, then SQL_WLONGVARCHAR is returned instead. |
| BIT VARYING | SQL_VARCHAR |

| PostgreSQL Type | SQL Type |
|---|---|
| | ■ If the length of the column is greater than the Max Varchar (`MaxVarchar`) setting, then SQL_LONGVARCHAR is returned instead.<br>■ If the Use Unicode option (the `UseUnicode` key) is enabled, then SQL_WVARCHAR is returned instead.<br>■ If the Use Unicode option (the `UseUnicode` key) is enabled and the column length is greater than the Max Varchar (`MaxVarchar`) setting, then SQL_WLONGVARCHAR is returned instead. |
| BOOLEAN | SQL_BIT<br>If the Show Boolean Column As String option (the `BoolsAsChar` key) is enabled, then SQL_VARCHAR is returned instead. |
| BOX | SQL_VARCHAR |
| BYTEA<br>(ESCAPE AND HEX FORMATS) | SQL_VARBINARY<br>If the Bytea As LongVarBinary option (the `ByteaAsLongVarBinary` key) is enabled, then SQL_LONGVARBINARY is returned instead. |
| CHAR | SQL_CHAR<br>■ If the length of the column is greater than the Max Varchar (`MaxVarchar`) setting, then SQL_LONGVARCHAR is returned instead.<br>■ If the Use Unicode option (the `UseUnicode` key) is enabled, then SQL_WCHAR is returned instead.<br>■ If the Use Unicode option (the `UseUnicode` key) is enabled and the column length is greater than the Max Varchar (`MaxVarchar`) setting, then SQL_WLONGVARCHAR is returned instead. |
| CID | SQL_VARCHAR |
| CIDR | SQL_VARCHAR |

| PostgreSQL Type | SQL Type |
|---|---|
| CIRCLE | SQL_VARCHAR |
| COMPOSITE TYPES | SQL_VARCHAR |
| DATE | SQL_TYPE_DATE<br>If you are using ODBC 2.0, the SQL type is SQL_DATE. |
| DATERANGE | SQL_VARCHAR |
| DECIMAL | SQL_NUMERIC |
| DOUBLE PRECISION | SQL_DOUBLE |
| ENUM | SQL_VARCHAR |
| FLOAT<br>(SAME AS DOUBLE PRECISION) | SQL_DOUBLE |
| GEOMETRY | SQL_VARCHAR |
| INT4RANGE | SQL_VARCHAR |
| INT8RANGE | SQL_VARCHAR |
| INET | SQL_VARCHAR |
| INTEGER | SQL_INTEGER |
| INTERVAL | SQL_VARCHAR |
| JSON | SQL_VARCHAR |
| JSONB | SQL_VARCHAR |
| LINE | SQL_VARCHAR |
| LSEG | SQL_VARCHAR |
| MACADDR | SQL_VARCHAR |
| MONEY | SQL_DECIMAL |
| NUMERIC | SQL_NUMERIC |
| NUMRANGE | SQL_VARCHAR |
| OID | SQL_INTEGER |
| PATH | SQL_VARCHAR |
| PG_LSN | SQL_VARCHAR |
| POINT | SQL_VARCHAR |
| POLYGON | SQL_VARCHAR |
| REAL | SQL_REAL |
| SERIAL | SQL_INTEGER |
| SMALLINT | SQL_SMALLINT |
| SMALLSERIAL | SQL_SMALLINT |
| TEXT | SQL_VARCHAR<br>■ If the Use Unicode option (the `UseUnicode` key) is enabled, then |

| PostgreSQL Type | SQL Type |
|---|---|
| | SQL_WVARCHAR is returned instead.<br>■ If the Text As LongVarChar option (the `TextAsLongVarchar` key) is enabled, then SQL_LONGVARCHAR is returned instead.<br>■ If both options are enabled, then SQL_WLONGVARCHAR is returned instead. |
| TID | SQL_VARCHAR |
| TIME (WITH TIME ZONE) | SQL_TYPE_TIME<br>If you are using ODBC 2.0, the SQL type is SQL_TIME. |
| TIME (WITHOUT TIME ZONE) | SQL_TYPE_TIME<br>If you are using ODBC 2.0, the SQL type is SQL_TIME. |
| TIMESTAMP | SQL_TYPE_TIMESTAMP<br>If you are using ODBC 2.0, the SQL type is SQL_TIMESTAMP. |
| TSQUERY | SQL_VARCHAR |
| TSRANGE | SQL_VARCHAR |
| TSTZRANGE | SQL_VARCHAR |
| TSVECTOR | SQL_VARCHAR |
| TXID_SNAPSHOT | SQL_VARCHAR |
| UUID | SQL_GUID |
| VARCHAR | SQL_VARCHAR<br>■ If the length of the column is greater than the Max Varchar (`MaxVarchar`) setting, then SQL_LONGVARCHAR is returned instead.<br>■ If the Use Unicode option (the `UseUnicode` key) is enabled, then SQL_WVARCHAR is returned instead.<br>■ If the Use Unicode option (the `UseUnicode` key) is enabled and the column length is greater than the Max Varchar (`MaxVarchar`) setting, then SQL_WLONGVARCHAR is returned instead. |

| PostgreSQL Type | SQL Type |
|---|---|
| XID | SQL_INTEGER |

# Security and Authentication

To protect data from unauthorized access, PostgreSQL data stores require all connections to be authenticated using user credentials. Some data stores also require connections to be made over the Secure Sockets Layer (SSL) protocol, either with or without one-way authentication. The Simba PostgreSQL ODBC Connector provides full support for these authentication protocols.

**Note:**

In this documentation, "SSL" refers to both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The connector supports TLS 1.0, 1.1, and 1.2. The SSL version used for the connection is the highest version that is supported by both the connector and the server.

The connector provides a mechanism that enables you to authenticate your connection using your PostgreSQL user name and password or the Kerberos protocol. For detailed configuration instructions, see Creating a Data Source Name in Windows or Creating a Data Source Name on a Non-Windows Machine.

Additionally, the connector supports SSL connections with or without one-way authentication. If the server has an SSL-enabled socket, then you can configure the connector to connect to it.

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

# Connector Configuration Properties

Connector Configuration Options lists the configuration options available in the Simba PostgreSQL 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 PostgreSQL ODBC Driver DSN Setup
- Additional Options
- Data Type Configuration
- SSL 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 PostgreSQL ODBC Connector, or via the key name when using a connection string or configuring a connection from a Linux or macOS computer:

- Allow Self-Signed Server Certificate
- Auth Type
- Authentication Mode
- Bytea As LongVarBinary
- Cache Size
- CheckCertificate Revocation
- Client Cert
- Client Key
- Custom SSL Certificate Path
- Database
- Minimum TLS
- Money As Decimal
- Password
- Port
- Proxy Password
- Proxy Port
- Proxy Server
- Proxy Username
- Retrieve Entire Result Into Memory
- Server

- Enable Proxy For PostgreSQL Connection
- Enable Read Only
- Enable Table Types
- Encrypt Password
- Enforce Single Statement
- Log Level
- Log Path
- Max Bytea
- Max LongVarChar
- Max Varchar
- Service Name
- Show Boolean Column As String
- Single Row Mode
- Text As LongVarChar
- Use Declare/Fetch
- Use System Trust Store
- Use Unicode
- User

# Allow Self-Signed Server Certificate

This option specifies whether the connector allows a connection to aPostgreSQL server that uses a self-signed certificate.

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

**Note:** This setting is applicable only when SSL is enabled and the system trust store is being used. For more information, see Use System Trust Store.

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

# Auth Type

| Key Name | Default Value | Required |
|---|---|---|
| `UseKerberos` | Standard (`0`) | No |

## Description

This option specifies whether the connector uses Kerberos authentication.

- Kerberos (`1`): The connector uses Kerberos authentication.
- Standard (`0`): The connector uses standard PostgreSQL user name and password authentication.

# Authentication Mode

The SSL certificate verification mode to use when connecting to PostgreSQL. The following values are possible:

- **verify-full**: Connect only using SSL, a trusted certificate authority, and a server name that matches the certificate.
- **verify-ca**: Connect only using SSL and a trusted certificate authority.
- **require**: Connect only using SSL.
- **prefer**: Connect using SSL if available. Otherwise, connect without using SSL.
- **allow**: By default, connect without using SSL. If the server requires SSL connections, then use SSL.
- **disable**: Connect without using SSL.

**Note:**
For information about SSL support in PostgreSQL, see "SSL Support" in the PostgreSQL Documentation: http://www.postgresql.org/docs/9.4/static/libpq-ssl.html.

| Key Name | Default Value | Required |
|---|---|---|
| `SSLMode` | `prefer` | No |

# Bytea As LongVarBinary

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

## Description

This option specifies the SQL data type that the connector uses to return Bytea data.

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

# Cache Size

The number of rows that the connector returns when Declare/Fetch Mode is enabled. For more information, see Use Declare/Fetch.

| Key Name | Default Value | Required |
|---|---|---|
| `Fetch` | `100` | Yes, if Declare/Fetch Mode is enabled. |

# CheckCertificate Revocation

This option specifies whether the connector checks to see if a certificate has been revoked while retrieving a certificate chain from the Windows Trust Store.

This option is only applicable if you are using a CA certificate from the Windows Trust Store (see Use System Trust Store).

- Enabled (`1`): The connector checks for certificate revocation while retrieving a certificate chain from the Windows Trust Store.
- Disabled (`0`): The connector does not check for certificate revocation while retrieving a certificate chain from the Windows Trust Store.

**Note:** This option is disabled when the `AllowSelfSignedServerCert` property is set to `1`. This option is only available in Windows.

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

# Client Cert

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

## Description

The full path of the file containing the client SSL certificate.

# Client Key

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

## Description

The full path of the file containing the secret key used for the client certificate.

# Custom SSL Certificate Path

The full path of the file containing the root certificate for verifying the server.

If this option is not set, then the connector looks in the folder that contains the connector DLL file.

| Key Name | Default Value | Required |
|---|---|---|
| `SSLCertPath` | The location of the connector DLL file. | No |

# Database

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

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

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

# Enable Proxy For PostgreSQL Connection

**Note:**
This option is used only when you configure proxy connections using the Additional Configuration dialog box.

This option specifies whether the connector passes the connection to PostgreSQL through a proxy server.

- Enabled: The connector passes the connection through a proxy server.
- Disabled: The connector does not pass the connection through a proxy server.

For information about configuring proxy connections, see Configuring Additional Options in Windows and Configuring a Proxy Connection on a Non-Windows Machine.

| Key Name | Default Value | Required |
|---|---|---|
| N/A | Clear | Yes, if using the Additional Configuration dialog box to configure a proxy connection. |

# Enable Read Only

This option controls whether the connector is in read-only mode.

- Enabled (`1`): The connection is in read-only mode, and cannot write to the data store.
- Disabled (`0`): The connection is not in read-only mode, and can write to the data store.

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

# Enable Table Types

This option specifies whether the connector recognizes table type information from the data source. By default, the connector only recognizes a single, generic table type.

- Enabled (`1`): The connector recognizes the following table types: TABLE, VIEW, SYSTEM TABLE, and LOCAL TEMPORARY.
- Disabled (`0`): All tables returned from the data source have the generic type TABLE.

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

# Encrypt Password

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

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

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

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

# Enforce Single Statement

This option specifies whether the connector returns SQL_ERROR immediately for any other queries that is executed if there is already an active query in execution under the same connection. The connector allows applications to allocate more than one statement handles and execute queries in each

statement handle concurrently per connection. However, the connector allows only one active statement at a time for each connection.

- Enabled (`1`): The connector allows one active query to be executed at a time. If there is already an active query in execution under the same connection, the connector returns SQL_ERROR immediately for any other queries that is executed if there is already an active query in execution under the same connection.
- Disabled (`0`): The connector allows more than one queries to be executed at a time, but all queries are still sent and executed sequentially. If there is already an active query in execution under the same connection, the connector blocks queries in other statement handles from execution until the active query finishes execution and retrieves all the data, or when the application calls SQLCloseCursor or SQLFreeHandle with a HandleType of SQL_HANDLE_STMT to indicate that the statement handle can be freed.

**Note:**

- If Enforce Single Statement and Use Multiple Statements are both enabled, Use Multiple Statements Mode takes precedence.
- The connector only allows multiple queries to be execute sequentially when the statement handles are allocated in different threads. If there is already an active query in execution under the same connection and the queries to be executed belong to statement handles that are allocated within the same thread, the connector returns SQL_ERROR immediately. For more information, see Query Processing Modes.

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

# 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:

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

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

| Key Name | Default Value | Required |
|---|---|---|
| LogLevel | OFF (0) | No |

# Log Path

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

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

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

# Max Bytea

| Key Name | Default Value | Required |
|---|---|---|
| MaxBytea | 255 | No |

## Description

The maximum data length for Bytea columns.

# Max LongVarChar

The maximum data length for LongVarChar columns.

- If the column is of type WVARCHAR, the length is in Unicode characters.
- Otherwise, the length is in UTF-8 code units.

| Key Name | Default Value | Required |
|---|---|---|
| MaxLongVarChar | 8190 | No |

# Max Varchar

The maximum data length for VARCHAR columns.

- If the column is of type WVARCHAR, the length is in Unicode characters.
- Otherwise, the length is in UTF-8 code units.

| Key Name | Default Value | Required |
|---|---|---|
| MaxVarchar | 255 | No |

# Minimum TLS

The minimum version of TLS/SSL that the connector allows the data store to use for encrypting connections. For example, if TLS 1.1 is specified, TLS 1.0 cannot be used to encrypt connections.

- TLS 1.0 (1.0): The connection must use at least TLS 1.0.
- TLS 1.1 (1.1): The connection must use at least TLS 1.1.
- TLS 1.2 (1.2): The connection must use at least TLS 1.2.

| Key Name | Default Value | Required |
|---|---|---|
| Min_TLS | TLS 1.2 (1.2) | No |

# Money As Decimal

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

## Description

This option specifies whether the connector removes non-decimal characters, such as the dollar sign ($) and thousands separator (,), when it returns data of type Money.

- Enabled (`1`): The connector removes non-decimal characters when it returns Money data.
- Disabled (`0`): The connector does not remove non-decimal characters.

# Password

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

| Key Name | Default Value | Required |
|---|---|---|
| `PWD`<br>OR<br>`Password` | None | Yes, if configuring the connector to use standard authentication. |

# Port

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

| Key Name | Default Value | Required |
|---|---|---|
| `Port` | `5432` | Yes |

# Proxy Password

The password that you use to access the proxy server.

| Key Name | Default Value | Required |
|---|---|---|
| `ProxyPwd` | None | Yes, if connecting to a proxy server that requires authentication. |

# Proxy Port

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

| Key Name | Default Value | Required |
|---|---|---|
| `ProxyPort` | None | Yes, if connecting through a proxy server. |

# Proxy Server

The host name or IP address of a proxy server that you want to connect through.

| Key Name | Default Value | Required |
|---|---|---|
| ProxyHost | None | Yes, if connecting through a proxy server. |

# Proxy Username

The user name that you use to access the proxy server.

| Key Name | Default Value | Required |
|---|---|---|
| ProxyUid | None | Yes, if connecting to a proxy server that requires authentication. |

# Retrieve Entire Result Into Memory

This option specifies whether the connector returns the entire query result into memory.

- Enabled (1): The connector returns the entire query result into memory.
- Disabled (0): The connector returns the query result in chunks or single rows.

When using keys to set connector options, you can enable this option by setting the SingleRowMode, UseDeclareFetch, and UseMultipleStatements keys to 0.

**Note:**
When using connection attributes to set connector options, you can enable this option by setting the SingleRowMode, UseDeclareFetch, and UseMultipleStatements attributes to 0.

| Key Name | Default Value | Required |
|---|---|---|
| N/A | Selected (1) | No |

# Server

A comma-delimited list of endpoint servers. The connector attempts to connect to each server in the order specified until it finds a valid server or the list has been exhausted. If a valid server cannot be found the connector alerts the user.

| Key Name | Default Value | Required |
|---|---|---|
| Server | None | Yes |

# Service Name

The Kerberos service principal name of the PostgreSQL server.

| Key Name | Default Value | Required |
|---|---|---|
| KerberosServiceName | postgres | No |

# Show Boolean Column As String

This option specifies the SQL data type that the connector uses to return Boolean data.

- Enabled (1): The connector returns Boolean columns as SQL_VARCHAR data with a length of 5.
- Disabled (0): The connector returns Boolean columns as SQL_BIT data.

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

# Single Row Mode

This option specifies whether the connector uses Single Row Mode and returns query results one row at a time. Enable this option if you plan to query large results and do not want to retrieve the entire result into memory.

- Enabled (1): The connector returns query results one row at a time.
- Disabled (0): The connector returns all query results at once.

When using connection attributes to set connector options, make note of the following:

- If SingleRowMode and UseDeclareFetch are both set to 0, then the connector retrieves the entire query result into memory.
- If UseDeclareFetch is set to 1, then it takes precedence over SingleRowMode.
- If SingleRowMode is set to 1 and UseDeclareFetch is set to 0, then SingleRowMode takes precedence over UseMultipleStatements.

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

# Text As LongVarChar

This option specifies the SQL data type that the connector uses to return Text data. The returned data type is also affected by the Use Unicode option (the UseUnicode key). For more information, see Use Unicode.

- Enabled (1): The connector returns Text columns as SQL_LONGVARCHAR data. If the Use Unicode option (the UseUnicode key) is also enabled, then the connector returns SQL_WLONGVARCHAR data instead.

- Disabled (0): The connector returns Text columns as SQL_VARCHAR data. If the Use Unicode option (the UseUnicode key) is also enabled, then the connector returns SQL_WVARCHAR data instead.

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

# Use Declare/Fetch

This option specifies whether the connector uses Declare/Fetch Mode and returns a specific number of rows at a time.

- Enabled (1): The connector uses Declare/Fetch Mode and returns a specific number of rows at a time. To specify the number of rows, configure the Cache Size option (the Fetch attribute).
- Disabled (0): The connector returns all rows at once.

When using keys to set connector options, make note of the following:

- If UseDeclareFetch is set to 1, then it takes precedence over SingleRowMode and UseMultipleStatements.
- If UseDeclareFetch is set to 0 and SingleRowMode is set to 1, then the connector returns query results one row at a time.
- If UseDeclareFetch and SingleRowMode are both set to 0, then the connector retrieves the entire query result into memory.

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

# Use Multiple Statements

This option specifies whether the connector can have more than one query, separated by a semicolon ( ; ), in a single SQLExecDirect call.

- Enabled (1): The connector can have more than one query, separated by semicolon ( ; ), in a single SQLExecDirect call. The connector returns all the query results into memory.
- Disabled (0): The connector executes one query at a time in SQLExecDirect.

When using connection attributes to set connector options, make note of the following:

- If UseDeclareFetch is set to 1, then it takes precedence over UseMultipleStatements.
- If UseDeclareFetch is set to 0 and SingleRowMode is set to 1, then SingleRowMode takes precedence over UseMultipleStatements.

| Key Name | Default Value | Required |
|---|---|---|
| UseMultipleStatements | Disabled (0) | No |

# Use System Trust Store

This option specifies whether to use a CA certificate from the system trust store, or from a specified .pem file.

- Enabled (1): The connector verifies the connection using a certificate in the system trust store.
- Disabled (0): The connector verifies the connection using a specified .pem file. For information about specifying a .pem file, see Custom SSL Certificate Path.

> **Note:** This option is only available in Windows.

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

# Use Unicode

This option specifies whether the connector returns PostgreSQL data as Unicode or regular SQL types.

- Enabled (1): The connector returns data as Unicode character types:
  - SQL_WCHAR is returned instead of SQL_CHAR.
  - SQL_WVARCHAR is returned instead of SQL_VARCHAR.
  - SQL_WLONGVARCHAR is returned instead of SQL_LONGVARCHAR.
- Disabled (0): The connector returns data as regular SQL types:
  - SQL_CHAR is returned instead of SQL_WCHAR.
  - SQL_VARCHAR is returned instead of SQL_WVARCHAR.
  - SQL_LONGVARCHAR is returned instead of SQL_WLONGVARCHAR.

For detailed information about how the connector returns PostgreSQL data as SQL types, see Data Types.

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

# User

The user name that you use to access the PostgreSQL server.

If you are using keys to set connector options, UID takes precedence over Username.

| Key Name | Default Value | Required |
|---|---|---|
| `UID`<br>OR<br>`Username` | None | Yes |

# Configuration Options Having Only Key Names

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

- ApplicationName
- ConnectionTimeout
- Driver
- KeepAlive
- KeepAliveCount
- KeepAliveInterval
- KeepAliveTime
- Locale

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

- UseLogPrefix

# ApplicationName

This property sets the name of the current application on the server. If not set, `ApplicationName` is set to the connector name and version upon connection.

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

# ConnectionTimeout

This property specifies the maximum wait time for the connection, in seconds. If set to `0` or not specified, the connector waits indefinitely. The minimum allowed value is `2`.

**Note:**
This property can also be configured via SQL_ATTR_LOGIN_TIMEOUT.

| Key Name | Default Value | Required |
|---|---|---|
| ConnectionTimeout | 0 | No |

# Driver

In Windows, the name of the installed connector for (`Simba PostgreSQL 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 PostgreSQL ODBC Connector` when installed in Windows, or the absolute path of the connector shared object file when installed on a non-Windows machine. | Yes |

# KeepAlive

When this option is enabled (`1`), the connector uses TCP keepalives to prevent connections from timing out.

When this option is disabled (`0`), the connector does not use TCP keepalives.

| Key Name | Default Value | Required |
|---|---|---|
| KeepAlive | 1 | No |

# KeepAliveCount

The number of TCP keepalive packets that can be lost before the connection is considered broken.

When this key is set to `0`, the connector uses the system default for this setting.

| Key Name | Default Value | Required |
|---|---|---|
| KeepAliveCount | 0 | No |

# KeepAliveInterval

The number of seconds between each TCP keepalive retransmission.

When this key is set to `0`, the connector uses the system default for this setting.

| Key Name | Default Value | Required |
|---|---|---|
| KeepAliveInterval | 0 | No |

# KeepAliveTime

The number of seconds of inactivity before the connector sends a TCP keepalive packet.

When this key is set to `0`, the connector uses the system default for this setting.

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

# Locale

The locale to use for error messages.

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

# UseLogPrefix

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

Set the property to one of the following values:

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

  For example, if you are connecting as a user named "jdoe" and using the connector in an application with process ID 7836, the generated log files would be named `jdoe_7836_simbapostgresqlodbcdriver.log` and `jdoe_7836_simbapostgresqlodbcdriver_connection_[Number].log`, where *[Number]* is a number that identifies each connection-specific log file.

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

  To configure this option for the Windows connector, you create a value for it in one of the following registry keys:
  - For a 32-bit connector installed on a 64-bit machine: **HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Simba\SimbaPostgreSQL ODBC Driver\Driver**
  - Otherwise: **HKEY_LOCAL_MACHINE\SOFTWARE\Simba\SimbaPostgreSQL ODBC Driver\Driver**

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

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

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

# Third-Party Trademarks

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

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

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

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

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

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

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

PostgreSQL is a trademark or registered trademark of The PostgreSQL Global Development Group or its subsidiaries in Canada, the United States and/or other countries.

All other trademarks are trademarks of their respective owners.