insightsoftware

Simba JDBC Bridge

Installation Guide

Version 1.2

July 2025

Copyright

Contact Us

www.insightsoftware.com

About This Guide

The *Simba JDBC Bridge Installation Guide* explains how to use Simba JDBC Bridge Installer and corresponding command lines to setup server as service and support JDBC access to a local ODBC driver. Simba JDBC Bridge is configured to use localhost as the listening address. It is essential that the client use localhost and the configured port to connect to the bridge. This guide is intended for end users of the Simba JDBC Bridge.

To use the Simba JDBC Bridge, the following knowledge is helpful:

- Familiarity with the platform on which you are using the Simba JDBC Bridge
- Ability to use the data store to which the Simba JDBC Bridge is connecting
- An understanding of the role of JDBC technologies in connecting to a data store
- Experience creating and configuring JDBC connections
- Exposure to SQL

Document Conventions

The following conventions are used throughout this guide to emphasize important concepts:

Italics are 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 blue exclamation mark indicates a short note appended to a paragraph.

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

Contents

Platform Support

Each machine where you use the Simba JDBC Bridge must have Java Runtime Environment (JRE) 11.0 installed.

The Simba JDBC Bridge supports Windows 64-bit and ODBC 64-bit drivers.

The connector complies with the JDBC 4.3 data standard.

Installer Setup

Start by downloading and extracting the package onto your computer. Then, run the installer for Simba JDBC Bridge.

To install the product using the setup wizard on Windows:

1. To start the setup wizard, double-click the `SimbaJDBCBridge.msi` file.
2. In the setup wizard, click **Next**.
3. Select the check box to accept the terms of the license agreement, and then click **Next.**
4. Choose the location on your computer where you want Simba JDBC Bridge to be installed:
   - To accept the default installation directory, click **Next**. By default, the installation location is `C:\Program Files\insightSoftware`
   - Or, to install the application to a different location, click **Change**, then type or browse to the location you want, and then click **OK**. To accept this installation directory, click **Next**.
5. Click **Install**. If you are prompted to allow the program to make changes to your computer, click **Yes**.

   When the installation is completed, click **Finish**.

You can now use Simba JDBC Bridge to create a service.

Bridge as a Service Setup

To create a bridge as a service in Windows, use the `sc` command.

The `sc` command is a command-line utility that allows users to manage Windows services, including creating, modifying, and deleting them.

For example, this is a basic syntax:

sc create [service name] [binPath=] [DisplayName=] [start=] [depend=] [obj=] [password=]

Parameters within square brackets are arguments for `sc create`. You can also configure Simba JDBC Bridge parameters, for more information, see Service Setup Parameters.

For example, a command using Simba JDBC Bridge parameters:

sc create SimbaBridge binpath= "C:\server\lib\SimbaBridge64.exe -LogPath C:\server\lib\log -LicenseFile C:\Users\Administrator\SimbaJDBCBridgeDriver.lic -ServerDSN SQLServer"

This command creates the SimbaBridge service with designated 64-bit binary path, Server DSN, log path, and license file path. If you would like the service to automatically start when Windows starts, use `start=auto`. For explanations of the settings of each parameter, see Service Setup Parameters.

To start the service use the following command:

sc start SimbaBridge

Now the service is running and the JDBC Client can be used to connect the configured ODBC driver.

To stop the service use the following command:

sc stop SimbaBridge

Note:
The service should be stopped before upgrading Simba JDBC Bridge.

To delete the service use the following command:

sc delete SimbaBridge

Note:
The service should be deleted before uninstalling.

Service Setup Parameters

The `sc create` command allows user to attach additional parameters to setup the service and Simba JDBC Bridge has its own specific parameters to configure the service.

Note:
Property values are case-sensitive.

LicenseFile

Default Value	Required	Allowed Value
Unspecified	No	Valid file path or unspecified.

Description

Specifies the directory where the license file is located.

Example:

-LicenseFile “C:\Users\Administrator\SimbaJDBCBridgeDriver.lic”

Defaults to checking `SimbaJDBCBridgeDriver.lic` in the current working directory of server. If configuring the license path fails or an expired license is used, the server starts but connections are rejected with a license error.

ListenPort

Default Value	Required	Allowed Value
`1543`	No	`1` to `65535` (TCP/IP port number range).

Description

The local port for SimbaServer to bind to.

Example:

-ListenPort 1583

ServerDSN is required for a successful connection. Simba JDBC Bridge does not start without proper DSN setup.

LogLevel

Default Value	Required	Allowed Value
`LOG_OFF`	No	See below.

Description

Use this parameter to control the granularity of the messages and events that are logged.

Example:

-LogLevel LOG_ERROR

Set to one of the following numbers:

- `0` or `LOG_OFF`: Disable all logging.
- `1` or `LOG_FATAL`: Enable logging on the FATAL level, which logs very severe error events that will lead the connector to abort.
- `2` or `LOG_ERROR`: Enable logging on the ERROR level, which logs error events that might still allow the connector to continue running.
- `3` or `LOG_WARNING`: Enable logging on the WARNING level, which logs events that might result in an error if action is not taken.
- `4` or `LOG_INFO`: Enable logging on method entry and exit points, and parameter values for debugging.
- `5` or `LOG_DEBUG`: Enable logging on the DEBUG level, which logs detailed information that is useful for debugging the connector.
- `6` or `LOG_TRACE`: Enable logging on all method entry points.

LogPath

Default Value	Required	Allowed Value
Unspecified	No	Valid directory path or unspecified.

Description

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

Example:

-LogPath “C:\Simba Technologies\Temp”

If this value is not set, the log files are written to the current working directory. When it's running as a service, the current working directory of the SimbaBridge process is: `C:\Windows\system32`.

ServerDSN

Default Value	Required	Allowed Value
None	Yes	Valid DSN string.

Description

The ODBC DSN used to connect to the bridged ODBC driver.

Example:

-ServerDSN SQLServer

ServerDSN is required for a successful connection. Simba JDBC Bridge does not start without proper DSN setup.

Other Service Setup Parameters

The `sc create` command has its own arguments that are not Simba JDBC Bridge specific to configure the service. There are many arguments and some common ones are listed here.

Note:

Property values are case-sensitive.

BinPath

Default Value	Required	Allowed Value
None	Yes	Valid path to SimbaBridge binary and command line options.

Description

The path to the service binary file.

Example:

binpath= "C:\server\lib\SimbaBridge64.exe -ServerDSN SQLServer”

There is no default value for this argument and this string must be supplied. The bin path should be pointing to where Simba JDBC Bridge is. The double quotes are needed if the parameter contains whitespace and the parameters for the bridge go in here.

Note:

Multiple service instances can be created by adding unique `-ListenPort [portnum]` values to the `binPath` and unique Service Name values.

DisplayName

Default Value	Required	Allowed Value
Unspecified	No	Valid informative string.

Description

The descriptive user-friendly name for the service that is displayed in the Services control panel.

Example:

displayname= "JDBCBridge"

Multiple data sources can be configured by setting up more than one service with different names, listen ports, and `ServerDSN`s.

Configuring Secure Sockets Layer (SSL)

SimbaClient/Server supports Secure Sockets Layer (SSL) encryption on the connection between SimbaClient and SimbaServer. If SSL is enabled, SimbaServer use OpenSSL to encrypt all data moving across the network connection.

Turning On SSL

To establish an SSL connection, you must set the required configuration properties on SimbaServer, as explained in Configuration Properties for SSL.

Using SSL Certificates

To configure SSL using certificates, you must generate a set of SSL certificates. The Certificate Authority (CA) certificate that is used to sign the Server Certificate becomes the `SslCACertfile` that the Client will use to authenticate the Server.

Example SSL certificates are included in the `[INSTALL_DIR]\SimbaEngineSDK\1\Documentation\SSLCertificates` directory. These are Simba self-signed certificates that were created using OpenSSL.

For more information, see Generating an SSL Certificate with Verisign or Generating a Certificate Authority (CA) Certificate for Self-Signing.

Finally, you must distribute the certificates. See Distributing SSL Certificates.

Using a Trusted Key Store

For the JDBC client, you can use either a trusted key store or SSL certificates, as explained in Creating a Trusted Key Store for JDBC Client.

Configuration Properties for SSL

To establish an SSL connection, you must configure the following properties on both SimbaClient and SimbaServer:

- `UseSSL` to specify that SSL is to be used.
- `SslCertfile` on the Server, or `SslCACertfile` on the client, to specify a certificate file.
- OR, `SslKeyFile` to specify a key file. Only one of `SslCertfile` or `SslKeyFile` needs to be specified.

To enable the OpenSSL FIPS module, set `EnableFIPS` to `1`. OpenSSL 3 must be properly configured when enabling FIPS, see Configuring OpenSSL FIPS.

UseSSL

The following table explains the type of connection that is established when `UseSsl` is set on the client and server:

	Client UseSsl= Disabled	Client UseSsl= Enabled	Client UseSsl= Required
Server UseSsl=Disabled	Connection. No SSL.	Connection. No SSL.	No Connection.
Server UseSsl=Enabled	Connection. No SSL.	Connection with SSL.	Connection with SSL.
Server UseSsl=Required	No Connection.	Connection with SSL.	Connection with SSL.

Example:

- When Simba Client has `UseSsl`set to `Enabled` and Simba Server has `UseSsl`set to `Required`, a connection using SSL will be established.
- When SimbaServer has `UseSSL` set to `Required`, SimbaClients that want to connect to it must have `UseSSL` set to either `Enabled` or `Required`.
- When SimbaServer has `UseSSL` set to `Enabled`, it will accept both SSL and non SSL connections from clients, though SSL will be used if the client has it set to either `Enabled` or `Required`.

EnableFIPS

Required	No
Default value	`0`
Comment	This property enables the OpenSSL FIPS module. Using this property affects both temporary swap file encryption and client/server communication. OpenSSL 3 must be properly configured when enabling FIPS.

Configuring OpenSSL FIPS

OpenSSL 3 includes FIPS support and by default it can be used without configuring OpenSSL. However, if FIPS support is enabled using the `EnableFIPS` configuration setting, the following must be configured in OpenSSL:

1. `openssl.cnf` and `fipsmodule.cnf` files must be configured.
    a. `openssl.cnf` is the general OpenSSL configuration file. Its name and location can be provided by the `OPENSSL_CONF` environment variable or by placing it in the default OpenSSL

install location on each platform. The default location can be determined by running the following command: `openssl version -d`

b. `fipsmodule.cnf` contains configuration and validation information specific to the fips module. This file need to be included with the `openssl.cnf` file with a `.include` directive. The SEN SDK provides a partial `fipsmodule.cnf` for each platform that can be used. The partial version of the file is portable but is not considered to be an installed version. When FIPS is properly installed, this file contains aditional lines about the install and the status of running the self tests. Without those lines, the FIPS module runs self tests each time it is loaded. To generate those lines, the `openssl fipsinstall` command must be run. For more details, see the OpenSSL documentation: https://www.openssl.org/docs/man3.0/man1/openssl-fipsinstall.html.

2. The `OPENSSL_MODULES` environment variable must point at the directory containing the fips module, such as `.dll` and `.so`.

For detailed information about configuring OpenSSL, see the OpenSSL documentation: https://www.openssl.org/docs/man3.0/man5/config.html.

Creating a Trusted Key Store for JDBC Client

For the JDBC Client, you can configure SSL using either an SSL certificate or a trusted key store.

To create a trusted key store:

1. Make sure that the Java `bin` directory is in your PATH variable.
2. Open a command window and run the following command:

   keytool -import -alias "JDBCKeyStore" -file *[Path]*\CA-cert.pem -keystore C:\JDBCKeyStore

   The TrustedStorePath keyword must point to `[Path]\JDBCKeyStore`.
3. You will be prompted for a password and be asked to verify that the given certificate should be trusted. Type **yes** when prompted.

Generating a Certificate Authority (CA) Certificate for Self-Signing

This section explains how to establish yourself as a root certificate authority for self-signing your certificates. Self-signed certificates are useful during development when you do not need to purchase a commercial certificate. For instructions on generating a commercially-signed certificate, see Generating an SSL Certificate with Verisign.

To install OpenSSL:

1. Install OpenSSL from http://openssl.org.
2. Add the path to the `openssl.exe` executable to your PATH variable. Refer to http://openssl.org for other configuration properties.

To create the root CA certificate:

1. Create the `C:\newcerts` directory:

   > md C:\newcerts

2. Change to the newcerts directory:

   > cd C:\newcerts

3. Generate a CA private key:

   > openssl genrsa -des3 -out CA-key.pem 2048

4. Generate the root CA certificate.

   > openssl req -new -key CA-key.pem -x509 -days 1000 -out CA-cert.pem

You will be prompted for information which will be incorporated into the certificate, such as Country, City, Company Name, etc. Remember what information you entered as you may get prompted for this information again at a later stage. When asked for an email address, provide the email address of the CA contact.

The root CA certificate is created.

You will need `CA-key.pem` and `CA-cert.pem` in the following steps.

To create a Signing a Server Certificate:

You will need `CA-key.pem` and `CA-cert.pem` from the previous step.

1. Generate a new key:

   openssl genrsa -des3 -out server-key.pem 2048

2. Generate a certificate signing request:

3. Locate the `openssl.cnf` file is in your OpenSSL installation directory.

4. Copy the `openssl.cnf` file to the `newcerts` directory. You may need to modify some of the configuration settings in this file.

5. Enter the following command (all in one line):

   openssl req -new -config openssl.cnf -key server-key.pem -out signingReq.csr

6. Self-sign the certificate using your `CA-cert.pem` certificate. Enter the following command (all in one line):

   openssl x509 -req -days 365 -in signingReq.csr -CA CA-cert.pem -CAkey CA-key.pem -CAcreateserial -out server-cert.pem

A server certificate is created and signed.

Generating an SSL Certificate with Verisign

This section explains how to generate an SSL Certificate with Verisign.

Ensure OpenSSL is installed, and the location of `openssl.exe` is added to your PATH variable. See Generating an SSL Certificate with Verisign.

To create the Server Private Key:

1. Create the directory `C:\newcerts`:

```
> md C:\newcerts
```

2. Change to the `newcerts` directory:

```
> cd C:\newcerts
```

3. Generate a new key:

```
openssl genrsa -out server-key.pem
```

4. Generate a certificate signing request:`openssl req -new -key server-key.pem -out signingReq.csr`

   You will be asked a series of questions which will be incorporated into the certificate request, such as Country, City, Company Name, etc. When asked for an email address, provide a valid email address because Verisign will send you the signed certificate via this email address.

   Note:

   The information you enter will be verified when you send the request to a trusted authority.

5. Send the request (`signingReq.csr`) to the Certificate Authority (Verisign). You may need to verify that the information collected when generating `signingReq.csr` is correct.

   If the request for certificate signing was successful, the Certificate Authority (Verisign) will send you a certificate using the email address you provided. In the email, there will be an encrypted CA certificate and a link to an encrypted CA intermediate certificate.

6. Copy both certificates to a text file, with the non-intermediate certificate followed by the intermediate certificate. This text file will be referred to as `CA-cert.pem` in the following steps.

To create and sign a Server Certificate:

1. Ensure you have the following files, which were generated in the previous sections:
   - `server-key.pem`
   - `signingReq.csr`
   - `CA-cert.pem`

2. Copy the `CA-cert.pem` file to your `C:\newcerts` directory. Ensure the `server-key.pem` and `signingReq.csr` files are in this directory as well.

3. Change to the newcerts directory.

```
> cd C:\newcerts
```

4. Create the server certificate. Enter the following command (all in one line):

```
openssl CA -in signingReq.csr -out server-cert.pem -keyfile server-key.pem -
days 365 -cert CA-cert.pem
```

The server certificate is created and signed.

Distributing SSL Certificates

In order to set up SSL, the following certificates are required:

Certificate Name	Description
Server key file, for example `server-key.pem`.	The file for the SSLKeyFile configuration keyword for SimbaServer.
Server certificate file, for example `server-cert.pem`.	The file for the SSLCertfile configuration keyword for SimbaServer.
Certificate authority file, for example `CA-cert.pem`.	The file for the SSLCACertFile configuration keyword for SimbaClient.

Configuring SimbaServer

SimbaServer configuration properties control functionality such as logging, security, and resource management. This section explains how end users can configure SimbaServer to meet their needs.

SimbaServer functionality is the same on Windows, Linux, Unix and macOS. The configuration properties are the same on all platforms. The only difference is where the configuration properties are stored.

Note:

- You can configure SimbaServer on the command line, through the Windows Registry (on Windows), or through configuration files (Linux, Unix, and macOS). Command-line settings take precedence.
- Logging properties cannot be set on the command line. To set logging properties, use the Windows Registry on Windows platforms or the `.ini` files on non-Windows platforms.

Updates to SimbaServer in Simba JDBC Bridge version 1.2

In the Simba JDBC Bridge version 1.2, SimbaServer was rewritten to improve performance and to be self-tuning. This allows it to maintain optimal performance while receiving large numbers of requests from the clients, while reducing system resources when client requests are minimal. As a result, the SimbaServer configuration is simplified and the number of configuration parameters are reduced.

Note:

If you are upgrading SimbaServer from an earlier version, your installer may want to map the existing configuration values at the customer's site to the new configuration values, where applicable.

Command Line Configuration

You can configure SimbaServer using the command line. If you set configuration properties on the command line, they take precedence over properties set in configuration files or in the Windows Registry.

Use the following format to set SimbaServer properties on the command line:

MyServer.exe -*[PROPERTY] [VALUE]* -*[PROPERTY] [VALUE]*

Example:

QuickJsonJNIDSIServer64.exe -ListenPort 1200

Configuring SimbaServer on Windows

On Windows, the configuration information is stored in the registry under the following key:

- ***[Registry_Root]*\SOFTWARE\Simba\Quickstart\Server**
- Or (for the 32-bit SimbaServer on a 64-bit machine)
  ***[Registry_Root]*\SOFTWARE\Wow6432Node\Simba\Quickstart\Server**

Where *Registry_Root* is one of the following;

- **HKEY_CURRENT_USER**
- Or, **HKEY_LOCAL_MACHINE** (recommended)

If you have configuration settings for SimbaServer under both keys, SimbaServer will stop looking after it finds any settings under the **HKEY_CURRENT_USER** key and will not look for any settings under the **HKEY_LOCAL_MACHINE** key.

We recommend that you create your configuration settings in the **HKEY_LOCAL_MACHINE** key for the following reasons:

- If you run SimbaServer as a Windows service, it will run by default under the System user ID. It is difficult to configure **HKEY_CURRENT_USER** registry values under the System user ID and while this difficulty can be solved in different ways, it is easier to avoid it completely. It is much easier to configure the values under the **HKEY_LOCAL_MACHINE** key, which is visible to all users.
- If your primary configuration is under the **HKEY_LOCAL_MACHINE** key, you can do testing under a user ID and create an overriding configuration in the **HKEY_CURRENT_USER** key of that user. This avoids having to change the **HKEY_LOCAL_MACHINE** key configuration until you know exactly what you want to do, and all other users IDs, including the System user ID, still see only the **HKEY_LOCAL_MACHINE** key configuration.

Configuring SimbaServer on Linux, Unix, and macOS

On Linux, Unix, and, macOS, the configuration information for servers is stored in a configuration file. By default, this file is named `simbaserver.ini` and is located in the directory containing the server executable.

You can also use an environment variable to override the name and location of the configuration file. By default, SimbaServer uses the configuration environment variable `SIMBAINI`. For example, if you set `SIMBAINI` to `/var/lib/mydata/AceDataConfig.ini`, SimbaServer will read configuration information from the file `/var/lib/mydata/AceDataConfig.ini` and ignore configuration information in the file `simbaserver.ini`.

Note:

- If the configuration environment variable is *set*, SimbaServer *ignores* the default configuration file. Customers can use the configuration environment variable to set the location of the configuration information at install time.
- If the configuration environment variable is *not set*, SimbaServer *uses* the default configuration file.

You can change the name of the default configuration file and the configuration environment variable.

simbaserver.ini format

The `simbaserver.ini` file contains the section name [Server]. This section contains the keyword and value for each set of options. The keyword and value are of the form `keyword=value`. The section ends with end of the file.

For example, set the logging level as follows:

Example:

```
[Server]

LogLevel=LOG_OFF

<eof>
```

SimbaServer Configuration Properties

See Configuring SimbaServer on Linux, Unix, and macOS for the location of these properties in the configuration files.

These properties can also be entered on the command line.

This table summarize the configuration properties. Detailed descriptions are provided in following tables.

Note:

For properties that list a maximum value of UINT_MAX, this equals a value of 2^{32}-1.

Keyword	Description
IdleTimeout	Connection idle timeout period.
ListenAddress	Bind the SimbaServer instance to a fixed IP address.
ListenPort	The local port for SimbaServer to bind to.
MaxConnections	The maximum number of connections allowed.
MaxWorkerThreads	Maximum number of active concurrent connections.
MinWorkerThreads	Number of Worker Threads created at startup.
ServerNameList	The hostname(s) or IP addresses of the machine where the SimbaServer is running.
ConnStmtLimit	Maximum number of simultaneous statements the server allows on any given connection.
	Logging configuration properties
LogLevel	Controls the granularity of the messages and events that are logged.

Keyword	Description
LogPath	Specifies the directory where the log files are created.
LogFileSize	The size of each log file. When the maximum size of the file is reached, a new file is created.
LogFileCount	The number of log files to create.
	Secure socket layer (SSL) properties
SslCertfile	The SSL certificate file to use with SSL secure connections.
SslKeyFile	The SSL private key file to use with SSL secure connections.
UseSsl	Enable SSL encryption for the connection between SimbaClient and SimbaServer.
	SimbaServerMain properties
Help	Windows only. Print a listing of the command-line options.
daemon	Not available on Windows. Instruct the server to run in the background.
StopEvent	Windows only. Specify a win32 event that can signal the server to shut down.

General Configuration Properties

ConnStmtLimit

Required	No
Range	0 - UINT_MAX Set to 0 for unlimited.
Default value	0 (No limit)
Example	`connstmtlimit=10`
Comment	The maximum number of simultaneous statements that the server allows on any given connection.

IdleTimeout

Specifies connection idle timeout period.

Required	No
Range	0 - UINT_MAX Set to 0 for no timeout.
Default value	86400 (24 hours)
Example	`IdleTimeout=86400`
Comment	The duration in seconds that a connection can remain idle, with no communication from a client, before SimbaServer disconnects it. Use this parameter to prevent network interruptions and other connection errors from consuming SimbaServer resources. **Note:** ■ The server IdleTimeout property and the ODBC client IdleTimeout property have the same name, but are different properties.

ListenAddress

Binds the SimbaServer instance to a fixed IP address.

Required	No
Range	Any valid IP address or hostname, or empty string.
Default value	Empty string.
Example	`ListenAddress=192.168.0.2`
Comment	This keyword restricts the server so it accepts TCP/IP connections only on the specified IP address. This is useful on machines with multiple IP addresses (for example, machines with multiple NICs), or to ensure that SimbaServer binds to the expected IP address. If `ListenAddress` is not specified, SimbaServer will do the following: Use `gethostname()` to get the fully qualified host name of the current machine. Use `getaddrinfo()` to resolve that name to an IP address. Bind to the resulting address.

ListenPort

The local port for SimbaServer to bind to.

Required	No

Range	0-65535 (TCP/IP port number range.)
Default value	1543
Example	`ListenPort=1583`
Comment	This keyword specifies the port to which the server will bind and listen for TCP/IP connection requests. The range of the value is 0-65535. If you do not set this value, SimbaServer will use the default port 1543. **Note:** The port `1543` is registered to Simba Technologies.

MaxConnections

Specifies the maximum number of total connections.

Required	No
Range	0 - UINT_MAX
Default value	512
Example	`MaxConnections=256`
Comment	This is the total number of connections that are permitted. Set to `0` for unlimited connections. `MaxConnections` includes both active and idle connections. Once this maximum number of connections is reached, subsequent connection requests will wait until one of the existing connections is disconnected. Refer to `MaxWorkerThreads` for the maximum number of active concurrent connections. If you expect a large number of users to access your server, this value should be set to a higher number.

MaxWorkerThreads

Specifies the maximum number of active concurrent connections.

Required	No
Range	0 - UINT_MAX
Default value	100
Example	`MaxWorkerThreads=32`

Comment	The maximum number of concurrent active connections that are permitted. Set to `0` for unlimited worker threads. Since each connection, when active, requires a worker thread to process its requests, the maximum number of worker threads is equal to the maximum number of active concurrent connections. When this maximum number is reached, an active connection will wait until one of the active connections has had its requests serviced. Its worker thread will then be freed for use by this active connection. **Note:** This property is NOT the maximum number of connections allowed - i.e. there can be concurrent idle connections. Idle connections do not require the use of a worker thread. If you expect a large number of concurrently active users, this number should be set higher. Refer to `MaxConnections` for the maximum total number of connections permitted.

MinWorkerThreads

Specifies the number of Worker Threads created at startup.

Required	No
Range	0-65535 worker threads
Default value	10
Example	`MinWorkerThreads=50`
Comment	The number of worker threads that SimbaServer will create before starting up the server. SimbaServer will not allow the number of worker threads in the thread pool to dip below this number. Each active connection uses one worker thread to process its requests.

ServerNameList

The hostname(s) or IP addresses of the machine where the SimbaServer is running.

Required	No, but recommended. If not supplied, SimbaServer will attempt to use DNS to discover the hostname of the machine on which it is running.
Default value	None

Example	`ServerNameList = Server1,Server2`
Comment	This property ensures the server can accurately communicate the hostname(s) and IP address(es) for the machine on which it is running. This avoids error in cases where more than one hostname is mapped to a single IP address, or the server is running behind a NAT-enabled firewall.

Logging Configuration Properties

Note:

Logging properties cannot be set on the command line. To set logging properties, use the Windows Registry on Windows platforms or the `.ini` files on non-Windows platforms.

LogLevel

Controls the granularity of the messages and events that are logged.

Required	No
Allowed values	See *Comment*.
Default value	`LOG_OFF`
Example	`LogLevel=LOG_ERROR`
Comment	With this keyword, you can control the amount of log output by controlling the kinds of events that are logged. Possible values (case sensitive): ■ `0` or `LOG_OFF`: no logging occurs ■ `1` or `LOG_FATAL`: only log fatal errors ■ `2` or `LOG_ERROR`: log all errors ■ `3` or `LOG_WARNING`: log all errors and warnings ■ `4` or `LOG_INFO`: log all errors, warnings, and informational messages ■ `5` or `LOG_DEBUG`: log method entry and exit points and parameter values for debugging ■ `6` or `LOG_TRACE`: log all method entry points

LogPath

Specifies the directory where the log files are created.

Required	No
Allowed values	Valid directory path, or unspecified.
Default value	Unspecified. This stores log files in the current working directory.
Example	`LogPath="C:\Simba Technologies\Temp"`
Comment	If this value is not set, the log files are written to the current working directory of the SimbaServer. **Note:** The current working directory for SimbaServer running as a service and SimbaServer running as an executable is different.

LogFileSize

Specifies the size, in bytes, of each log file.

Note:

You can use `LogFileSize` and `LogFileCount` to enable automatic cleanup of log files. When one log file reaches the size specified by `LogFileSize`, a new log file is created. When the number of log files reaches the limit specified by `LogFileCount`, the first log file is deleted when another log file is created.

Required	No
Allowed values	The Simba JDBC Bridge will accept any positive integer. The maximum size of a file depends on the host machine's specifications.
Default value	20971520 bytes
Example	`LogFileSize="30000000"`
Comment	When the maximum size of the log file is reached, another log file will be created.

LogFileCount

Specifies the number of log files to create.

Required	No
Allowed values	The Simba JDBC Bridge will accept any positive integer. The maximum number of files depends on the host machine's specifications.
Default value	50

Example	`LogFileCount=100`
Comment	When the maximum number of log files has been created, the oldest file will be deleted and a new one created.

SSL Configuration Properties

For more information on configuring SSL, Configuring Secure Sockets Layer (SSL).

UseSsl

This setting allows the connection between the SimbaClients and SimbaServer to use Secure Sockets Layer (SSL) encryption.

Note:

You must configure SSL on both the SimbaServer and the Simba JDBC Bridge

Required	Yes, if `UseSsl` is enabled on the client.
Allowed values	`Disabled`, `Enabled`, `Required`
Default value	`Disabled`
Example	`UseSsl=Enabled`

SslCertfile

Specifies the SSL certificate file to use with SSL secure connections.

Required	Yes, when `UseSsl`is `Enabled` or `Required`.
Data type	String
Range	Valid absolute or relative directory path to the SSL certificate file.
Default value	None
Example	`SslCertFile=C:\SampleServerCertificate.pem`
Comment	This keyword specifies the full or relative path to the SSL certificate file to use with SSL secure connections. Note that the server will use this information only when you turn on SSL security with the UseSsl keyword.

SslKeyFile

Specifies the SSL private key file to use with SSL secure connections.

Required	Yes, when `UseSsl` is `Enabled` or `Required`.
Data type	String
Range	Valid absolute or relative directory path to the SSL server key file.
Default value	None
Example	`SslKeyFile=C:\SampleServerKey.pem`
Comment	This keyword specifies the full or relative path to the SSL private key file to use with SSL secure connections. Note that the server will use this information only when you turn on SSL security with the `UseSsl` keyword.

SimbaServerMain properties

These properties are available on the command line for SimbaServers that use the `SimbaServerMain` library. They are not available in a configuration file. If you include SimbaServer in your own process and implement your own `main()` function, these properties are not available.

Help

List and describe the command-line options.

Required	No

Example	`QuickstartDSIIServer.exe -Help`
	`QuickstartDSIIServer.exe /Help`

Comment	This parameter does not take any arguments. The server will not start up when this parameter is used, and will ignore all other arguments.

daemon

Instruct the server to run as a daemon (in the background without a terminal window).

Required	No
Example	`QuickstartDSIIServer.so -daemon`
Comment	This parameter is not available on Windows.

StopEvent

Specify an event object that can signal the server to shut down.

Required	No

Default Value	By default, no event is monitored.
Example	`QuickstartDSIIServer.exe -StopEvent NoLicenseAvailable`
Comment	This parameter is available on Windows only. For more information on event objects, see https://msdn.microsoft.com/en-ca/library/windows/desktop/ms682655%28v=vs.85%29.aspx.

ReportListenAddresses

The file to write the listen addresses to, relative or absolute.

Required	No
Default Value	N/A.
Example	`C:\[install dir]\logs\`
Comment	If specified, the server will write the address or addresses it is listening on to the specified file. The file will have the following format: *[IP] [Port]*\\n\"

Configuring SimbaClient for JDBC

SimbaClient for JDBC configuration properties control logging, server discovery, and security. Configuration properties are either added to the connection URL or implemented programmatically in the JDBC application.

Connection URL

The following is the connection URL format for the JDBC client:

jdbc:simba://localhost:[port];[property]=[value];[property]=[value]

For example:

jdbc:simba://localhost:1543;UID=BartonL;PWD=sneaky;LogLevel=0;LogPath=C:\Simba Technologies\Temp

Linking to the connector class

You must link the JDBC application to the correct connector class:

For JDBC 4.3 connectors:

com.simba.client.core.jdbc43.SCJDBC43DataSource

SimbaClient for JDBC Configuration Properties

All configuration properties are configured either programmatically or on the connection string. The following table summarizes the configuration properties.

Note:
For properties that list a maximum value of UINT_MAX, this equals a value of $2^{32}-1$.

Property	Description
Timeout Properties:	
`LoginTimeout`	The time to wait for a response during login.
`ConnectionTimeout`	The time to wait for a response from the server.
Logging configuration properties:	
`LogLevel`	Controls the granularity of the messages and events that are logged.

Property	Description
`LogPath`	Specifies the directory where the log files are created.
Other:	
`D2OJDBCFunctionCatalogMode`	Controls the behavior of DatabaseMetadata getProcedures or getProcedureColumns and getFunctions or getFunctionColumns for JDBC clients.

General configuration properties

Along with the specific properties that apply to the client itself, any other properties are passed along to the ODBC driver, in the connection string used to connect to it.

The following are general configuration properties.

Note:
Property values are case-sensitive.

LoginTimeout

Default Value	Required	Allowed Value
`60`	No	`0` to UINT_MAX seconds.

Description

The time, in seconds, to wait for a response from the server after a login request is made by the client.

Example:

LoginTimeout=10

A value of `0` means no timeout. The value of this property is used to set the value of SQL_ATTR_LOGIN_TIMEOUT. A log in timeout may occur earlier than the time specified by this value. For example, if a DNS lookup failure occurs, the log in attempt times out immediately.

ConnectionTimeout

Default Value	Required	Allowed Value
`0`	No	`0` to UINT_MAX seconds.

Description

The time, in seconds, to wait for a response from the server after a login request is made by the client.

Example:

ConnectionTimeout=10

A value of `0` means no timeout.

LogLevel

Default Value	Required	Allowed Value
`OFF`	No	See below.

Description

Use this parameter to control the granularity of the messages and events that are logged.

Example:

LogLevel=ERROR

> **Note:**
>
> Log files are not created if this value is set to `OFF`.

Set to one of the following values:

- `OFF`: Disable all logging.
- `FATAL`: Enable logging on the FATAL level, which logs very severe error events that will lead the connector to abort.
- `ERROR`: Enable logging on the ERROR level, which logs error events that might still allow the connector to continue running.
- `WARNING`: Enable logging on the WARNING level, which logs events that might result in an error if action is not taken.
- `INFO`: Enable logging on the INFO level, which logs general information that describes the progress of the connector.
- `DEBUG`: Enable logging on method entry and exit points, and parameter values for debugging.
- `TRACE`: Enable logging on all method entry points.

LogPath

Default Value	Required	Allowed Value
Unspecified	No	Valid directory path or unspecified.

Description

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

Example:

LogPath=C:\Simba Technologies\Temp

If this value is not set, the log files are written to the current directory of the JVM using the client.

D2OJDBCFunctionCatalogMode

Default Value	Required	Allowed Value
`NotSupported`	No	See below.

Description

Controls the behaviour of DatabaseMetadata.getProcedures or getProcedureColumns and getFunctions or getFunctionColumns for JDBC clients.

Example:

D2OJDBCFunctionCatalogMode=EmulateFunctionCatalogs

Set to one of the following numbers:

- `NotSupported`: The default value, if not specified. Methods getFunctions or getFunctionColumns returns an error.
- `NoFunctions`: getFunctions or getFunctionColumns returns an empty result and no error.
- `EmulateFunctionCatalogs`: SimbaBridge emulates getFunctions or getFunctionColumns by splitting the results of the underlying ODBC driver's SQLProcedures or SQLProcedureColumns between getProcedures or getProcedureColumns and getFunctions or getFunctionColumns. Information for procedures for which SQLProcedures returns SQL_PT_FUNCTION in the PROCEDURE_TYPE column is returned by getFunctions or getFunctionColumns, and information for all other procedures is returned by getProcedures or getProcedureColumns.

Third-Party Trademarks