

Magnitude Simba SDK

Simba Client/Server Developer Guide

Version 10.1.17

November 2019

About This Guide

Purpose

This guide explains how to use Simba SDK to connect ODBC and JDBC applications to remote data stores in networked environments.

Simba SDK includes networking components that enable communication over TCP/IP between client and server machines. Client machines host the ODBC or JDBC connector, and server machines host a server component containing your Data Store interface implementation (DSII). You can use the DSII that you developed for a stand-alone connector and simply recompile it as a server.

Audience

The guide is intended for developers who have created a connector with the Simba SDK, and who want to use Simba SDK to enable distributed deployments of the connector. This guide is also intended for end users of the Simba SDK.

Knowledge Prerequisites

To use the Simba SDK, the following knowledge is helpful:

- Familiarity with the platform on which you are using the Simba SDK.
- Ability to use the data store to which the Simba SDK is connecting.
- An understanding of the role of ODBC or JDBC technologies and driver managers in connecting to a data store.
- Experience creating and configuring ODBC or JDBC connections.

Document Conventions

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.

i NOTE:

Indicates a short note appended to a paragraph.

⚠ IMPORTANT:

Indicates an important comment related to the preceding paragraph.

Variables Used in this Document

The following variables are used in this document:

Variable	Description
<i>[DRIVER_NAME]</i>	The name of your connector, as used in Windows registry keys and names of configuration files.
<i>[INSTALL_DIR]</i>	Installation directory for the Simba SDK.

Table of Contents

About This Guide	2
Variables Used in this Document	3
Introducing Simba Client/Server	6
Deployment Options	6
The Modern Server-based DBMS	6
SimbaServer Solutions	7
SimbaClient/Server Architecture	7
Example: ODBC Client/Server Deployment	10
Build the Quickstart Connector as a SimbaServer	10
Configure the Server	12
Configure a DSN for the SimbaClient	13
Test the Client/Server Deployment	14
Working with Simba Client/Server	16
Development Strategy	16
Test Strategy	16
Building SimbaServer	17
Building SimbaServer on Windows	18
Building SimbaServer on Linux, Unix, and macOS	19
Configure SimbaServer as a Windows Service	20
Troubleshooting	21
Installing SimbaClient/Server	22
Gather the Required Files	22
SimbaServer Required Files	22
SimbaClient for ODBC Required Files	24
SimbaClient for JDBC Required Files	24
Visual C++ ODBC Redistributable Files	25
Installing SimbaServer on Windows	25
Installing SimbaClient for ODBC	26
Installing SimbaClient for JDBC	26
Testing the Client/Server Connection	27

Configuring SimbaServer	28
Command Line Configuration	28
Configuring SimbaServer on Windows	29
Configuring SimbaServer on Linux, Unix, and macOS	31
SimbaServer Configuration Properties	33
Auto-Reconnect (ODBC only)	45
Configuring SimbaClient for ODBC	47
Configuring SimbaClient for ODBC on Windows	47
Configuring SimbaClient for ODBC on Linux, Unix, and macOS	49
SimbaClient for ODBC Configuration Properties	51
Configuring SimbaClient for JDBC	66
Connection URL	66
Linking to the connector class	66
SimbaClient for JDBC Configuration Properties	66
Configuring Secure Sockets Layer (SSL)	75
Turning On SSL	75
Using SSL Certificates	75
Using a Trusted Key Store	75
Configuration Properties for SSL	75
Creating a Trusted Key Store for JDBC Client	76
Generating a Certificate Authority (CA) Certificate for Self-Signing	77
Generating an SSL Certificate with Verisign	78
Distributing SSL Certificates	79
Setting Properties to Control Logging	81
Example: Logging Properties for the QuickStart SimbaServer and ODBC Client	81
Contact Us	83
Kerberos Authentication Support	85
Example: Configuring Kerberos for C++ Servers	86
Example: Configuring Kerberos For Java DSII Connectors	91
Configuration Properties for Integrated Security	97
Frequently Asked Questions	99
Third-Party Trademarks	101

Introducing Simba Client/Server

SimbaServer turns your Data Store interface implementation (DSII) into a full-featured, remote Relational Database Management System (RDBMS). You can use the same DSI implementation that you developed for a stand-alone connector, and simply link in the SimbaServer libraries to create a database server. Your new database server can be accessed by SimbaClients from any platform supported by the Simba SDK.

The SimbaClient uses the same Simba components as used in your stand-alone connector, so your customers will see the same behavior whether they use the stand-alone connector or a client/server solution.

Deployment Options

Both ODBC and JDBC clients are supported. The core SimbaServer code is implemented in C++. While you can develop your DSII in Java or DotNet, it must connect to the C++ server core. You can develop your DSII as follows:

- C++ SimbaServer SDK.
- Java SimbaServer SDK with a JNI bridge.
- DotNet SimbaServer SDK with Common Language Infrastructure (CLI).

You can deploy SimbaClient and SimbaServer on different platforms. For example, you can deploy SimbaServer on Linux and SimbaClient on Windows. Or, you could deploy the client on 32-bit Windows and the server on 64-bit Windows.

SimbaServer can also be loaded as a shared library, allowing you to embed it within your own application.

For more information on supported platforms, see *Platform and Compiler Requirements* in *Simba SDK Developing Connectors for SQL-capable Data Stores* or *Simba SDK Developing Connectors for Data Stores Without SQL*.

The Modern Server-based DBMS

Modern Relational Database Management Systems (RDBMSs) provide access for users via a remote network protocol that runs on common networks such as TCP/IP. This provides nearly universal access to the RDBMS since a well-designed database protocol will run on most networks, and virtually all user machines are already networked. All major commercial RDBMSs work this way.

The figure below illustrates the flexibility of deployment with a server-based RDBMS:



Database access via a remote protocol also introduces tremendous flexibility in the choice of deployment architectures, because a remote network protocol creates an interface that is independent of language,

operating system, word length, processor, and network. A well-designed remote protocol allows any two machines to communicate. As a result, clients and servers can be deployed where they make the most sense to your customers: servers can be deployed on high-powered, highly reliable machines, while clients can be deployed to maximize user convenience.

SimbaServer Solutions

SimbaServer is most frequently used as a stand-alone executable, although it can be set up as a DLL or shared object under another server. You must link SimbaServer to a DSI implementation before it can be an executable. The DSI implementation can include Simba SQL Engine or not, and it can be written to perform a wide range of functionality including SQL query processing with Simba SQL Engine, concentrating client requests through one executable, aggregating data stores, or controlling data access through role-based permissions. Likewise, a SimbaServer written in Java should include the JNI Server. There are many possibilities for using SimbaServer as an intermediate processing step in a larger system.

SimbaClient/Server Architecture

The architecture of a complete Simba Client/Server solution is very similar to that of a stand-alone connector. All of the same functionality is present, with the addition of the client/server functionality that transports the DSI functionality across the network. The following diagram compares the stand-alone Simba SQL Engine ODBC connector and an equivalent client/server solution:



The ODBC application and the driver manager are the same in both ODBC stacks.

Note:

There is only one driver manager in the Simba Client ODBC stack. Some ODBC client architectures have two driver managers in the stack, which introduces the problem of keeping them synchronized.

The top end of SimbaClient for ODBC uses the same SimbaODBC components as the top end of the stand-alone connector. As a result, their response to ODBC function calls is the same.

The section of code that connects the ODBC functionality to the DSI API is the only place where the two stacks differ:

- In the stand-alone connector, the SimbaODBC layer connects directly to the DSI API.
- In the client/server stack, the client/server connection mechanism connects the SimbaODBC code to the DSI API.

The ODBC client component and the server component handle the network communication, effectively projecting the DSI API across the network.

SimbaClient/Server Deployment Options

SimbaClient/Server is a collection of smaller components that allow remote access to your data store. SimbaServer is most frequently used as a stand-alone executable, although it can be set up as a DLL or shared object under another server. You must link SimbaServer to a DSI implementation in order to create an executable. The DSI implementation can include Simba SQLEngine or not, and it can be written to perform a wide range of functionality including SQL query processing with Simba SQLEngine, concentrating client requests through one executable, aggregating data stores, or controlling data access through role-based permissions. Likewise, a SimbaServer written in Java should include the JNI Server. There are many possibilities for using SimbaServer as an intermediate processing step in a larger system.

Running SimbaClient and SimbaServer

When you start up your linked SimbaServer, it binds to a configurable port on your server machine and listens for connection requests from SimbaClient connectors. Since all SimbaClient connector use the same protocol, they are all handled in the same way by SimbaServer. When a SimbaClient connector finds SimbaServer, it requests a connection. With a successful connection, the SimbaClient and SimbaServer begin a conversation using the SimbaClient/Server protocol. This is a layered protocol designed for clients making remote data queries and optimized for transmitting the result sets back to the client. It is independent of the standard interface used by the user application.

SimbaServer is designed to optimize use of shared server resources, while SimbaClient is designed to optimize the responsiveness of the application to give the best experience to the user. The protocol parameters are configurable in case the default parameters do not provide the best performance for your circumstances.

Reusing your DSI Implementation

One of the important design features of SimbaServer is that it links downward to exactly the same DSI implementation as SimbaODBC, which enables it to serve as a simpler development environment than SimbaServer. This means that you can first develop and test your DSI implementation as a local stand-alone ODBC connector using SimbaODBC. This is a simpler initial environment than developing using SimbaServer. When your new DSI implementation is performing to your satisfaction, you can link it to SimbaServer and begin testing it in a remote way. If you know the state of the logic and performance of the DSI implementation before introducing client/server, you can reduce your investigation time and debugging costs.

SimbaEngine contains a SimbaClient for ODBC and a SimbaClient for JDBC that provide direct access to SimbaServer.

SimbaClient for ODBC

SimbaClient for ODBC is an ODBC connector DLL or shared object that can connect to SimbaServer. It includes SimbaODBC and a DSI implementation that communicates via the Simba Client/Server protocol to SimbaServer. Since any SQL Engine in the stack will be on the server side, there is no need for Simba SQL Engine in this connector. This is a completely generic ODBC connector that, when queried, reports the capabilities of the database that is connected to SimbaServer.

You do not need to modify the ODBC connector.

SimbaClient for JDBC

SimbaClient for JDBC is a JDBC connector packaged as a Jar file so you can install it in an end user's client-side Java Run Time Environment. SimbaClient for JDBC includes the equivalent of SimbaODBC and custom Java code that communicates via the Simba Client/Server protocol with SimbaServer.

You do not need to modify the JDBC connector.

Example: ODBC Client/Server Deployment

This example demonstrates how build the C++ Quickstart sample connector as a SimbaServer, then deploy it with the ODBC client in a client/server configuration on a single Windows machine. You can use this information to understand the basic principles of ODBC client/server deployment, then use the information in the rest of this guide to deploy your own connector in a distributed deployment (with the client and server components on different machines), on the platform of your choice.

Build the Quickstart Connector as a SimbaServer

You can rebuild the Quickstart or Ultralight sample as a SimbaServer. You can also rebuild your own DSII as a SimbaServer. The SimbaServer contains the DSI implementation (DSII) and will handle accessing the data store.

The following instructions use the Quickstart connector as an example, but you can also use the Ultralight connector.

To build the Quickstart connector as a SimbaServer on Windows:

1. Open the VisualStudio project for Quickstart (for example, `QuickstartDSII_vs2013.sln`, depending on the version of Visual Studio) at `[INSTALL_DIR]\SimbaEngineSDK\10.1\Examples\Source\Quickstart\Source`.
2. Select the bitness (Win32 or x64), then select a server build configuration, for example:
 - `Debug_Server`
 - Or, `Debug_MTDLL_Server`
3. Build the project.

The SimbaServer executable, `QuickstartDSIIserver<BITS>.exe`, is built. By default the executable is built to the following location:

```
[INSTALL_
DIR]\SimbaEngineSDK\
10.1\Examples\Source\Quickstart\Bin\<<BUILD>\<RELEASE|DEBUG><CONFIGURATION>,
where
```

- `<BUILD>` is a combination of your operating system, machine bitness, and compiler
- `<RELEASE|DEBUG>` is `release` or `debug`
- `<CONFIGURATION>` is `mt` if you select MTDLL as the solution configuration, otherwise `md`

For example:

```
C:\Simba
Technologies\SimbaEngineSDK\10.1\Examples\Source\Quickstart\Bin\Windows_
vs2013\debug32md\QuickstartDSIIserver32.dll
```

To build the Quickstart connector as a SimbaServer on Unix and Linux:

The sample connectors included with the Simba SDK are installed in the folder `[INSTALL_DIR]/SimbaEngineSDK/10.1/Examples`. The sample connectors include sample makefiles.

In the following instructions, replace `[INSTALL_DIR]` with the Simba SDK installation directory, for example `/Library/Simba_XCode7`.

1. Set the **SIMBAENGINE_DIR** environment variable:
`export SIMBAENGINE_DIR=[INSTALL_DIR]/SimbaEngineSDK/10.1/DataAccessComponents`
2. Set the **SIMBAENGINE_THIRDPARTY_DIR** environment variable:
`export SIMBAENGINE_THIRDPARTY_DIR=[INSTALL_DIR]/SimbaEngineSDK/10.1/DataAccessComponents/ThirdParty`
3. Change to the following directory:
`[INSTALL_DIR]/SimbaEngineSDK/10.1/Examples/Source/Quickstart/Source`
4. Type `./mk.sh MODE=debug BUILDSERVER=1` to run the makefile for the debug server target.

This script calls the sample makefile, which automatically detects the required settings based on your operating system, machine bitness, and compiler.

The resulting library, `QuickstartServer<BITNESS>.so`, is put in the following directory:

```
[INSTALL_DIR]
/SimbaEngineSDK/
10.1/Examples/Source/Quickstart/Bin/<BUILD>/<RELEASE|DEBUG><BITNESS>
```

Where:

- `<BUILD>` is a combination of your operating system, machine bitness, and compiler
- `<RELEASE|DEBUG>` is either `release` or `debug`
- `BITNESS` is 32, 64, or 3264.

Example: Unix and Linux

```
[INSTALL_DIR]/SimbaEngineSDK/10.1/Examples/Source/Quickstart/Bin/Linux_x86_
gcc/debug64/QuickstartServer.so
```

Example: macOS

`[INSTALL_DIR]/SimbaEngineSDK/10.1/Examples/Source/Quickstart/Bin/Darwin_x86_Xcode7/debug64/QuickstartServer64.dylib`

Configure the Server

Set the configuration properties as described in the table below. This configures your SimbaServer for communication with the SimbaClient on the same machine.

Use one of the following Windows registry keys to set server configuration properties:

- **HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Simba\Quickstart\Server** for 32-bit servers on 64-bit machines
- OR, **HKEY_LOCAL_MACHINE\SOFTWARE\Simba\Quickstart\Server** for all other deployments.

Property	Value	Description
DBF	<code>[INSTALL_DIR]\SimbaEngineSDK\10.1\Examples\Databases\Quickstart</code>	Path to the text files used as a data store by the Quickstart connector. Note: DBF is only used by the Quickstart connector. Your server does not need this value to be set.
DriverLocale	en-US	Set the connector locale to US English.
ErrorMessagePath	<code>[INSTALL_DIR]\SimbaEngineSDK\10.1\DataAccessComponents\ErrorMessage</code>	Error messages are read from this directory.
ListenAddress	:::1	In this example, the client and server are installed on the same machine, so the ListenAddress is set to localhost. (By default, IPv6 is used so localhost is defined as :::1.)

Property	Value	Description
ListenPort	1543 (or any unused port)	Specify the port on which your new SimbaServer will listen to requests from the SimbaClient.

Once the server properties are configured, you can start the server.

To start the SimbaServer:

- Double-click `QuickstartDSIIserver.exe`.

Configure a DSN for the SimbaClient

The SimbaODBCClientDSII DSN is installed with the Simba SDK. We will modify this DSN to work with the sample SimbaServer. When you are ready to configure your own DSN:

- You can use the SimbaODBCClientDSII DSN as a starting point, or you can create your own from scratch.
- We recommend that you use the ODBC Data Sources UI to configure the DSN, rather than editing the registry directly.
- See [Configuring SimbaClient for ODBC](#) on page 47 for all configuration properties.

To modify the SimbaODBCClientDSII DSN:

1. Open **Administrative Tools > Data Sources (ODBC)**, then select the **System DSN** tab.
2. Select **SimbaODBCClientDSII** then click **Configure**.
3. Enter the following information:

Property	Description
Data Source Name	Enter a name for the data source. Your customers will see this name, but it does not have to match another setting.
Description	Enter a DSN description. Your customers will see this description, but it does not have to match another setting.
Server IP	::1 (This defines localhost for IPv6)

Property	Description
Server Port	1543 (this must match the server's ListenPort)
Idle Timeout	0
Login Timeout	60
Query Timeout	0

4. Click **OK**, and then click **OK**.

The SimbaClient DSN is configured. You can see the configuration information in the Windows registry at:

- **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\[Data Source Name]**
- Or, **HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBC.INI\[Data Source Name]** for 32-bit connectors on 64-bit machines.

Note:

The ODBC Data Sources tool sets some additional configuration values.

Test the Client/Server Deployment



While you can use any ODBC application to test your deployment, this section explains how to test the deployment using **ODBC Test**, which is included with the Microsoft Data Access Components Software Development Kit. You can download this software from <https://www.microsoft.com/en-ca/download/details.aspx?id=21995>.

Important:

The bitness of your ODBC Test application must match the bitness of the SimbaClient. You may have to search for **ODBC Test** in the **Windows programs and files** search bar in order to find the correct application. (For example, Windows may hide the 32-bit ODBC Test application on 64-bit machines).

To test the client/server deployment:

1. Make sure that the SimbaServer executable is running.
2. Launch the ODBCTest application that matches the bitness of your SimbaClient.
3. Click to request a Full Connect.

4. Click the DSN you configured in [Configure a DSN for the SimbaClient](#) on page 13. For example, SimbaODBCClientDSII.
5. Click **OK**. The connection is established and a message such as "Successfully connected to DSN 'SimbaODBCClientDSII'" appears.
6. Execute a SQL statement:
 - a. In the statement window (at the top), type `SELECT * FROM EMP.`
 - b. Select  and  to fetch and display the data.

The retrieved data is displayed, for example:



You have successfully built SimbaServer, configured the SimbaClient DSN, and deployed SimbaClient and SimbaServer on the same Windows machine.

Working with Simba Client/Server

To simplify development and make it easier to find and fix errors, we recommend an iterative approach to developing Simba Client/Server.

Development Strategy

To build your own Client/Server solution, we recommend you follow the steps below:

1. Build the server version of the SimbaEngine Quickstart Connector example, as described in [Example: ODBC Client/Server Deployment](#) on page 10.
2. Build your stand-alone ODBC connector and test it for correctness and reliability.
3. Rebuild your DSI implementation into a server, and test the client-server deployment. For more information, see [Building SimbaServer](#) on page 17.

Test Strategy

Because the stand-alone connector and SimbaServer with the same DSI implementation are so similar, you can use both of them tactically to reduce your testing complexity. Test the stand-alone connector first, and debug and fix the problems you find there before testing the SimbaServer version. This ensures that your DSI implementation is correct and reliable before you introduce the complexity of the client/server components.

Another consideration is that a stand-alone connector built with your DSI implementation should deliver exactly the same results as the SimbaServer version. Any differences point to an underlying problem that you should investigate.

Building SimbaServer

The ODBC and JDBC clients can both use the same server. The clients are shipped by Simba Technologies Inc., so you do not need to compile them.

To create a server from your DSI implementation, use the project files or make files included in the Simba SDK to link your C++ DSII code with the server libraries.

Note:

The core SimbaServer code is implemented in C++. While you can develop your DSII in Java or DotNet, you must connect to the C++ server core. You can develop your DSII by using the following:

- C++ SimbaServer SDK.
- Java SimbaServer SDK with a JNI bridge.
- DotNet SimbaServer SDK with Common Language Infrastructure (CLI).

The server libraries are described below:

Windows library	Linux, Unix, and macOS library	Description
SimbaServer.lib	libSimbaServer.a	Library containing the server functions.
CSCCommon.lib	libSimbaCSCCommon.a	Library containing code for client interaction.

Windows library	Linux, Unix, and macOS library	Description
SimbaServerMain.lib	libSimbaServerMain.a	<p>Optional. Library containing SimbaServer <code>main()</code> function.</p> <p>Include this library if you want to use Simba's <code>main()</code> function to create a stand-alone server executable. If you want to link SimbaServer in with your own process, you do not need this library.</p> <div style="border: 1px solid #ccc; padding: 10px; background-color: #f9f9f9;"> <p>Note:</p> <p>For information on linking SimbaServer with your own process, contact Simba support.</p> </div>

If you link with the SimbaServerMain library, SimbaServer will build as an executable (* .exe on Windows). On Windows and Linux, Unix, and macOS you can run this executable from a user's command line. This makes it easy to start and stop the executable for testing.

Note:

When building the solution, keep the following best practices in mind:

- Build the solution as a server
- Clean the solution before building the solution

Building SimbaServer on Windows

In Visual Studio, use a `server` build configuration to build your connector, for example `Debug_Server` or `Debug_MTDLL_Server`.

Note:

On Windows, you can also run the server executable as a service.

Building SimbaServer on Linux, Unix, and macOS

To build SimbaServer on Linux, Unix, and macOS platforms, set the environment variable `BUILDSEVER` to `exe`. The sample makefile included with the Simba SDK supports building a SimbaServer.

[Building an ODBC Connector as a Server](#)

Configure SimbaServer as a Windows Service

You can run SimbaServer as a Windows service, and configure it to start automatically when Windows starts. Your customers may prefer this configuration because your custom JDBC or ODBC connector starts automatically when the machine restarts.

Note:

- Run all commands from a command line that has administrative privileges or is "Run as Administrator" (for Windows Vista or later).
- These steps use the Quickstart connector as an example. Change "Quickstart" to the name of your connector.

To configure the server to run as a service:

1. Ensure your `DSIDriverFactory()` implementation calls `Simba::Server::SetServiceName()` with the name of the service. In the Quickstart sample connector, the service name is "SimbaQuickstartService".
2. Ensure the required DLLs are available on the machine where you are running the service, as described in [SimbaServer Required Files](#) on page 22.
3. Open a command line with administrative privileges.
4. Navigate to the directory that contains `QuickstartDSIIIServer.exe`.
5. Run the following from the command line:

```
sc create SimbaQuickstartService binpath= "<Full Path To Quickstart Server>\QuickstartDSIIIServer32.exe"
```

The SimbaQuickstartService is created.

Tip:

Your installer can configure the server to run as a service by executing the command line described above.

6. View the new service in the Windows Services:



To start and stop the service using the command line:

1. To start the service, execute the following from a command prompt:

```
net start SimbaQuickstartService
```

2. To stop the service, execute the following from a command prompt: `net stop SimbaQuickstartService:`

To uninstall the service:

- Execute `QuickstartDSIIIServer.exe -Uninstall` from a command prompt, or have your installer execute the same command line.

Troubleshooting

If your service will not start:

- Ensure the required dependencies are included on the machine where the server is running. For more information about required dependencies, see [Gather the Required Files](#) on page 22.
- Ensure you have used the correct service name in the command `sc create <Service Name>`. This name must exactly match the service name specified in the call to `SimbaSettingReader::SetServiceName()`.

Installing SimbaClient/Server

This section describes how to create an installer to deploy SimbaServer and the SimbaClient(s) at your customer's site.

Installation on Windows typically requires an installer for the server executable and client libraries, followed by client and server configuration. Installation on Linux, Unix, and macOS typically requires shipping a `.tar.gz` file and a list of instructions.

The rest of this section assumes that you want to create an installer or installation package for your client and server.

Gather the Required Files

The first step in developing your installation, whether creating an automatic installer or simply a `.tar.gz` file, is to list and gather all the required files. This includes the following:

- SimbaServer required files, as described in [SimbaServer Required Files](#) on page 22
- SimbaClient required files, as described in [SimbaClient for ODBC Required Files](#) on page 24 or [SimbaClient for JDBC Required Files](#) on page 24.
- Error message files.
- Localization files.

Important:

Some of the files for the server installation and the SimbaClient will be the same, particularly the ICU translation libraries, the error messages and possibly the localization files. The same files may be included in two different installers.

SimbaServer Required Files

SimbaServer Required Files for Windows

You need the following files to ensure that SimbaServer will work properly when installed on Windows at your customer's site:

Windows File	Description
The server executable (.exe file).	Your new server executable with your DSI implementation. For example, the Quickstart executable is named QuickstartDSIIServer.exe.
sbicudt53_32.dll for 32-bit or sbicudt53_64.dll for 64-bit	ICU utility libraries.
sbicuin53_32.dll for 32-bit or sbicuin53_64.dll for 64-bit	These libraries are in a subdirectory of
sbicuuc53_32.dll for 32-bit or sbicuuc53_64.dll for 64-bit	[INSTALL_DIR\SimbaEngineSDK\10.1\ DataAccessComponents\ThirdParty
libeay32.dll	SSL utility libraries.
ssleay32.dll	These libraries are in a subdirectory of
	[INSTALL_DIR\SimbaEngineSDK\10.1\ DataAccessComponents\ThirdParty
Visual C++ ODBC redistributable files	See Visual C++ ODBC Redistributable Files on page 25.

SimbaServer Required Files for Linux, Unix, and macOS

To ensure that SimbaServer will work properly when installed on Linux at your customer's site, include all the .so files from this directory:

```
[INSTALL_
DIR]
/SimbaEngineSDK/
10.1/DataAccessComponents/ThirdParty/icu/53.1.x/<BUILD>/<RELEASE|DEBUG>/lib
```

Where:

- **<BUILD>** is a combination of your operating system, machine bitness, and compiler
- **<RELEASE|DEBUG>** is either `release` or `debug`
- **BITNESS** is 32, 64, or 3264.

For example, on a Linux machine, using the GNU compiler and the release32 target, include all the .so files from the following directory:

[INSTALL_
DIR]/SimbaEngineSDK/10.1/DataAccessComponents/ThirdParty/icu/53.1.x/Linux_
x86_gcc/release32/lib

SimbaClient for ODBC Required Files

You need the following files to ensure that SimbaClient for ODBC works properly when installed at your customer's site:

Windows File	Description
SimbaClient.dll	The standard SimbaClient ODBC connector.
SimbaClientConfig.cfg	The ODBC configuration dialog. (Windows only).
SimbaClientConnectionDialog.dll	The SQL connection configuration dialog. (Windows only).
sbicudt53_32.dll for 32-bit or sbicudt53_64.dll for 64-bit sbicuin53_32.dll for 32-bit or sbicuin53_64.dll for 64-bit sbicuuc53_32.dll for 32-bit or sbicuuc53_64.dll for 64-bit	ICU utility libraries.
libeay32.dll ssleay32.dll	SSL utility libraries.
ClientMessages.xml ODBCMessages.xml	Error messages.
Visual C++ ODBC redistributable files	See Visual C++ ODBC Redistributable Files on page 25.

SimbaClient for JDBC Required Files

The JDBC client is deployed as a single JAR file, which is included with the Simba SDK. A different jar file is provided for each version of JDBC. You will need the correct file to ensure that SimbaClient for JDBC will work properly when installed at your customer's site:

- `SimbaJDBCClient_Release42.jar` supports JDBC 4.2
- `SimbaJDBCClient_Release41.jar` supports JDBC 4.1
- `SimbaJDBCClient_Release4.jar` supports JDBC 4.0

Visual C++ ODBC Redistributable Files

SimbaClient for ODBC and SimbaServer required the Visual C++ redistributables. These system files are distributed by Microsoft and are required by ODBC connectors and connections. Some Windows machines may have these files already installed, but in some cases the installer must install them.

Note:

If you have trouble connecting to ODBC data sources, ensure the Visual C++ ODBC redistributable files are installed.

Microsoft licenses these files to be distributed and redistributed free of charge. You may install them on as many machines as you want with no restriction.

You can download the package from Microsoft: <https://support.microsoft.com/en-ca/help/2977003/the-latest-supported-visual-c-downloads>.

Installing SimbaServer on Windows

When you have gathered all the required files described in [Gather the Required Files](#) on page 22, you can start to build your installer. This section describes the steps that the installer must perform.

To install SimbaServer:

1. Determine where you will install SimbaServer on the target machine. We recommend installing in its own folder to prevent file collisions.
2. Copy all of the SimbaServer required files, listed in [SimbaServer Required Files](#) on page 22, to the target folder. Most of these files will be used from this folder.
3. Create the registry entries for the configuration, as described in [Configuring SimbaServer](#) on page 28.
4. Start SimbaServer from the command line or by using the **Start -> Run command**.

Productizing Your Connector in Simba SDK Developing Connectors for SQL-capable Data Stores or Simba SDK Developing Connectors for Data Stores Without SQL.

Installing SimbaClient for ODBC

You can install SimbaClient for ODBC either on the same machine as SimbaServer or on a different machine. If you install both components on the same machine, the network software simply loops back to the SimbaServer process and connect normally. This is a simpler setup for development and test than using two machines.

To install SimbaClient for ODBC:

1. Determine where to install SimbaClient for ODBC on the target machine. We recommend installing in its own folder to prevent file collisions.
2. Copy all of the SimbaClient required files, listed in [SimbaClient for ODBC Required Files](#) on page 24, to the target folder.
3. On Windows, create the required registry entries. On other platforms, create the equivalent configuration files.

For example registry settings, see the following files in `[INSTALL_DIR]\SimbaEngineSDK\10.1\Documentation\Setup`:

- `SetupSimbaClient-32on32.reg`
- `SetupSimbaClient-32on64.reg`
- `SetupSimbaClient-64on64.reg`

Productizing Your Connector in Simba SDK Developing Connectors for SQL-capable Data Stores or Simba SDK Developing Connectors for Data Stores Without SQL.

Installing SimbaClient for JDBC

You can install SimbaClient for JDBC either on the same machine as SimbaServer or on a different machine. If you install both on the same machine, the network software will simply loop back to the SimbaServer process and connect normally. This is a simpler setup for development and test than using two machines.

To install SimbaClient for JDBC:

1. Copy the JDBC client JAR file, `SimbaJDBCClient_Release[version].jar`, to the required location on the client machine. The JAR file must be in the classpath of the JDBC application.
2. Determine where you will install SimbaClient for JDBC on the target machine. It needs to be accessible from the classpath of the JDBC application that will use it.
3. Copy the required JAR file or files, listed in [SimbaClient for JDBC Required Files](#) on page 24, to the target folder.

The JDBC application will need to be configured to use the SimbaClient for JDBC.

Testing the Client/Server Connection

If you have successfully installed and started SimbaServer, you can test the connection from the client to the server using a sample ODBC or a sample JDBC application. For an example of using a sample ODBC application, see [Test the Client/Server Deployment](#) on page 14.

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 and 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 SDK version 10.0

In the Simba SDK version 10.0, 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]`

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.

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]\SOFTWAREWow6432Node\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 for DotNet

If you are building the connector using the DotNet framework, you will also need to take the following steps to configure the connector:

1. Use GAC to add the `DotNetQuickstartDSII.dll`.
2. In the registry, check the following settings:
 - Under `ODBC.INI > SimbaODBCClientDSII`
 - Under `Simba > DotNewQuickstart > Server`
3. Start the server by running `QuickstartCLIDSIserver64.exe` in the build folder.
4. 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 you are using.

5. Find the `SimbaODBCClientDSII` DSN, and click **Configure**.
6. Click **Options > Add**.
7. In the **Key** field, type `DBF`
8. In the **Value** field, type the path to where the database is stored.
9. Click **OK > OK**.
10. To test the connection, click **Test**. Review the results as needed, and then click **OK**.

Rebranding configuration subkeys

To rebrand your version of the `SimbaServer`, you can change the location of server configuration properties in the Windows Registry. By default, `SimbaServer` uses the following subkey for its configuration settings:

- `SOFTWARE\Simba\Quickstart\Server`
- Or (for the 32-bit `SimbaServer` on a 64-bit machine)
`SOFTWARE\Wow6432Node\Simba\Quickstart\Server`

You can change `Simba\Quickstart` to your own company and connector name by using `SimbaSettingReader::SetConfigurationBranding()`.

Example:

Suppose you want to set the subkey for configuration settings to the following:

`HKEY_LOCAL_MACHINE\SOFTWARE\AceData\MyDriver`

You would use the following method call:

```
SimbaSettingReader::SetConfigurationBranding("AceData\\MyDriver")
```

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.

Renaming the default configuration file with `SetConfigurationBranding()`

By default, SimbaServer looks for configuration information in a file named `simbaserver.ini` in the directory containing the server executable.

Note:

If customers set a value for the configuration environment variable before starting SimbaServer, the default configuration file is ignored.

To change the name and location of the default configuration file, use:

```
SimbaSettingReader::SetConfigurationBranding([arg1])
```

where `[arg1]` is either:

- The path and file name of the configuration file.
- Or, the file name of the configuration file.

If the path is not specified, SimbaServer will look in the directory containing the server executable.

You can specify the following values for the path:

- \$HOME

SimbaServer will look for the configuration information in the user's home directory. It expects the file name to be preceded with a period (.).

For example, calling `SimbaSettingReader::SetConfigurationBranding("$HOME\my_driver.ini")`

tells SimbaServer to look in the user's home directory for the file `.my_driver.ini`

Note:

SimbaServer will expect the configuration file in the user's home directory to start with a period, for example `.my_driver.ini`, even though you did not specify a period in the file name.

- \$ETC

SimbaServer will look for the configuration information in the `/etc` directory. (It does *not* expect the file name to be preceded with a period (.)).

- Any valid path, for example `/var/lib/mydata/`.

Example:

To tell SimbaServer to look for configuration information in the file `/opt/AceData/lib/my_driver.ini`:

```
SimbaSettingReader::SetConfigurationBranding  
("/opt/AceData/lib/my_driver.ini");
```

To tell SimbaServer to look for configuration information in the file `my_driver.ini` in the default directory:

```
SimbaSettingReader::SetConfigurationBranding  
("my_driver.ini");
```

To tell SimbaServer to look for configuration information in the file `.my_driver.ini` in the user's home directory:

```
SimbaSettingReader::SetConfigurationBranding  
("$HOME\my_driver.ini");
```

Renaming the configuration environment variable with `SetUnixConfigEnvVariable()`

By default, SimbaServer looks in the environment variable `SIMBAINI` to find the name and location of the configuration file. To change the name of this environment variable, use:

```
SimbaSettingReader::SetUnixConfigEnvVariable([arg1])
```


where *[arg1]* is the name of the environment variable.

Example

Suppose you configure both the default configuration file and the configuration environment variable:

```
SimbaSettingReader::SetConfigurationBranding("/opt/AceData/lib/my_driver.ini");  
SimbaSettingReader::SetUnixConfigEnvVariable("AceDataConfig");
```

If customers install your SimbaServer and do *not* set the `AceDataConfig` environment variable, then SimbaServer reads configuration information from `/opt/AceData/lib/my_driver.ini`.

Later, suppose customers set the environment variable `AceDataConfig`:

```
export AceDataConfig="/usr/etc/acedata.ini"
```

After restarting SimbaServer, configuration information is read from `/usr/etc/acedata.ini`.

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 Windows](#) on page 29 for the location of these properties in the Windows registry, or [Configuring SimbaServer on Linux, Unix, and macOS](#) on page 31 for the location of these properties in the configuration files.

These properties can also be entered on the command line.

Note:

As of Simba SDK version 10.0, all configuration properties are stored in one section called *Server*. In previous releases, the configuration properties were divided into the sections *Admin*, *Buffers*, *Network* and *Threads*.

This table summarizes 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.
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.

Keyword	Description
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	<code>connstmtlimit=10</code>
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	<code>IdleTimeout=86400</code>
Comment	<p>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.</p> <p>Note:</p> <ul style="list-style-type: none">• The server <code>IdleTimeout</code> property and the ODBC client <code>IdleTimeout</code> on page 56 property have the same name, but are different properties.• For more information on timeout behavior, see How does timeout work? on page 99

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	<code>ListenAddress=192.168.0.2</code>

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.

Comment 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`

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.

Comment 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	<code>MaxConnections=256</code>
Comment	<p>This is the total number of connections that are permitted. Set to 0 for unlimited connections.</p> <p><code>MaxConnections</code> 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 <code>MaxWorkerThreads</code> 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.</p>

MaxWorkerThreads

Specifies the maximum number of active concurrent connections.

Required	No
Range	0 - UINT_MAX
Default value	100
Example	<code>MaxWorkerThreads=32</code>

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.

Comment

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	<code>MinWorkerThreads=50</code>
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	<code>ServerNameList = Server1,Server2</code>
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

For more information on logging, see [Setting Properties to Control Logging](#) on page 81.

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 <i>Comment</i> .
Default value	<code>LOG_OFF</code>

Example `LogLevel=LOG_ERROR`

With this keyword, you can control the amount of log output by controlling the kinds of events that are logged.

Possible values (case sensitive):

Comment

- 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"`

If this value is not set, the log files are written to the current working directory of the SimbaServer.

Comment

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 SDK will accept any positive integer. The maximum size of a file depends on the host machine's specifications.
Default value	20971520 bytes
Example	<code>LogFileSize="30000000"</code>
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 SDK will accept any positive integer. The maximum number of files depends on the host machine's specifications.
Default value	50
Example	<code>LogFileCount=100</code>
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, see [Configuring Secure Sockets Layer \(SSL\)](#) on page 75.

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 SDK

Required	Yes, if <code>UseSsl</code> is enabled on the client.
Allowed values	<code>Disabled</code> , <code>Enabled</code> , <code>Required</code>
Default value	<code>Disabled</code>
Example	<code>UseSsl=Enabled</code>

SslCertfile

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

Required	Yes, when <code>UseSsl</code> is <code>Enabled</code> or <code>Required</code> .
Data type	String
Range	Valid absolute or relative directory path to the SSL certificate file.
Default value	None
Example	<code>SslCertFile=C:\SampleServerCertificate.pem</code>
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 <code>UseSsl</code> keyword.

SslKeyFile

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

Required	Yes, when <code>UseSsl</code> is <code>Enabled</code> or <code>Required</code> .
Data type	String
Range	Valid absolute or relative directory path to the SSL server key file.
Default value	None
Example	<code>SslKeyFile=C:\SampleServerKey.pem</code>
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 <code>UseSsl</code> 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	<code>QuickstartDSIIIServer.exe -Help</code> <code>QuickstartDSIIIServer.exe /Help</code>
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	<code>QuickstartDSIIIServer.so -daemon</code>

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 `QuickstartDSIIIServer.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\n"`

Auto-Reconnect (ODBC only)

You can configure SimbaServer to automatically reconnect to ODBC clients. This feature is only supported where both the client and server are version 10.1.16 or later.

If any of the following conditions are true, the connector does not attempt to automatically reconnect a socket that was forcibly closed:

- The connection is currently in the midst of a transaction.
- The socket currently has an active request on it. For this purpose, *active* means there is at least one request which has been fully sent and either the request is not idempotent, or some but not all responses for the request have been received by the client.
- Any statements on the connection are in the midst of an execution. This is distinct from the point above since the client can appear idle when waiting for the application to provide some DATA-AT-EXEC parameter data still in execution.
- Any statements on the connection have open cursors for which there still exists data to fetch from the server, or there exist subsequent unfetched results for the current execution. There is no way to ensure that any query issued is either idempotent or would return results in the same order. This includes cursors for catalog functions such as SQLTables.
- Any statements are prepared.

To enable the auto-reconnect feature, the following configuration options must be set for the server via `Simba::DSI::ServerSemantics`:

- `Simba::DSI::ServerSemantics::GetDSIIType()`
Assigns a GUID that uniquely identifies the DSII. This is not set by default, and must be set to use Auto-Reconnect.
- `Simba::DSI::ServerSemantics::GetDSIIVersion()`
Retrieves a string that denotes the DSII version. This information must be identical on the new server for auto-reconnect to succeed. Returns the value of the `DSI_DRIVER_VER` connector property by default.
- `Simba::DSI::ServerSemantics::GetDSIIConfig()`
Retrieves a string that encodes the configuration of the DSII. Examples of this information could include the DBF location, or the underlying ODBC connector type (for D2O), or the back-end server being connected to. This information must be identical on the server being connected to for auto-reconnect to succeed.

Configuring SimbaClient for ODBC

SimbaClient configuration properties control logging, server discovery, and security. Configuration properties are the same on all supported platforms, but are configured in different ways as described in the following sections.

For information on individual configuration properties, see [SimbaClient for ODBC Configuration Properties](#) on page 51.

Configuring SimbaClient for ODBC on Windows

On Windows, the SimbaClient configuration properties are stored in the Windows registry. There are three registry locations where the different types of properties are stored:

Property Type	Location
DSN (Properties for configuring the DSN, or connection string).	<p>HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBC.INI\<i>[DSN name]</i> for 32-bit clients on 64-bit machines</p> <p>Or, HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\<i>[DSN name]</i></p> <p>Note: DSN properties can also be stored under HKEY_CURRENT_USER.</p>
Driver (Properties for configuring the connector binary).	<p>HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBCINST.INI\<i>[Driver name]</i> for 32-bit clients on 64-bit machines</p> <p>Or, HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\<i>[driver name]</i></p> <p>Note: Driver properties cannot be stored under HKEY_CURRENT_USER.</p>

Property Type	Location
Simba Setting Reader (Properties for configuring logging).	<p>HKEY_LOCAL_MACHINE\SOFTWARE\ Wow6432Node\Simba\SimbaClient\Driver for 32-bit clients on 64-bit machines</p> <p>Or, HKEY_LOCAL_MACHINE\SOFTWARE\Simba\SimbaClient\Driver</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>Note:</p> <p>Simba Setting Reader properties cannot be stored under HKEY_CURRENT_USER.</p> </div>

Note:

You cannot change the location of ODBC client logging properties in the Windows Registry.

HKEY_LOCAL_MACHINE vs HKEY_CURRENT_USER

Properties stored under **HKEY_LOCAL_MACHINE** are system-wide and are visible to all users on the machine. Properties stored under **HKEY_CURRENT_USER** are user-specific, so they are visible to the user currently logged in to the machine. Driver and Simba Setting Reader properties are located under **HKEY_LOCAL_MACHINE**. DSN properties can be located under **HKEY_LOCAL_MACHINE** or **HKEY_CURRENT_USER**.

Note:

Logging information is system-wide. All users on the same machine will have the same logging level.

Using the configuration dialog

The Simba ODBC client connector, like most ODBC connectors, has a configuration DLL that allows you to configure DSNs with a dialog rather than directly editing the Windows registry. Using the configuration dialog is safer than editing the Windows registry because the logic in the dialog prevents you from saving an inconsistent configuration. We recommend that you use the SimbaClient for ODBC configuration dialog.

To use the SimbaClient for ODBC configuration dialog:

1. Ensure SimbaClient for ODBC is properly installed on your system.
2. Click **Start > Control Panel > Administrative Tools > Data Sources (ODBC)** to open the ODBC Data Source Administrator.
3. Select type of DSN you want to configure: **User DSN, System DSN, or File DSN.**
4. Do one of the following:
 5. • Click on **Add...** to create a new data source, then select **SimbaODBCClientDSII** from the list of ODBC connectors.
 - Or, select a DSN configured to use SimbaClient for ODBC, then click on **Configure...** .
6. The configuration dialog opens.
7. On the main tab of the configuration dialog, you will see several of the DSN configuration properties such as Server IP and Server Port. All SimbaClient DSNs need these entries so they can connect to SimbaServer.
8. If you click **Option, Logging** and **Secondary Servers**, you will be able to see all of the other configuration properties used in the DSN.
9. To add a configuration keyword that does not have a corresponding UI element, click **Options** and add the keyword to the **Custom Property** list.

Note:

Configuration properties that not have a corresponding UI element in the configuration dialog are unlikely to be required by customers, but can be added by developers using the **Custom Property** list.

10. Click **OK**.

The configuration properties are written to the registry.

Configuring SimbaClient for ODBC on Linux, Unix, and macOS

On Linux, Unix, and macOS, the SimbaClient configuration properties are stored in configuration files. There are three files where the different types of properties are stored:

Property Type	Location
DSN	<code>odbc.ini</code> file
Driver	<code>odbcinst.ini</code> file

Property Type	Location
Simba Setting Reader	<code>simbaclient.ini</code> file

DSN properties

The DSN definition is stored in a text file called `odbc.ini`. The driver manager that you are using on the SimbaClient machine typically looks in the following locations for this file:

1. `$ODBCSYSINI/odbc.ini`
(Note that there is no leading period ".").
2. `$HOME/.odbc.ini`
>(Note the leading period "." in the name).
3. `/etc/odbc.ini`
(Note that there is no leading period ".").

The driver manager passes the configuration settings from this file to the SimbaClient.

Important:

Different driver managers may search different locations for the `odbc.ini` file. Use the correct location for your chosen driver manager.

To configure the DSN, use a text editor to edit the `odbc.ini` (there is no configuration dialog for Linux, Unix, and macOS).

In the `odbc.ini` file, each DSN has its own section that lists its configuration properties in the form `keyword=value`. Each section starts with the DSN name in square brackets and ends with the title of the next DSN or with the end of the file. For example:

```
[SimbaODBCClientDSII]
Description=Sample 32-bit SimbaEngine SimbaODBCClient DSII
Driver=[INSTALLDIR]SimbaEngineSDK/10.1/DataAccessComponents/Bin/Linux_
x86/libSimbaClient.so
Locale=en-US
SSLCACertFile=
[INSTALLDIR]SimbaEngineSDK/
10.1/Documentation/SSLCertificates/SampleCACertificate.pem
UseSsl=0
ServerList=127.0.0.1 1543
```

SimbaClient for ODBC Configuration Properties

All configuration properties are type String. See [Configuring SimbaClient for ODBC on Windows](#) on page 47 for the location of these properties in the Windows registry, or [Configuring SimbaClient for ODBC on Linux, Unix, and macOS](#) on page 49 for the location of these properties in the configuration files.

The following table summarizes the configuration properties. Detailed descriptions are provided in subsequent tables.

Note:

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

Property	Type	Description
ConnectionDialog	DSN, Driver, or SimbaSettingReader	The location of the connection dialog file.
Description	DSN	A brief, human-readable description of the DSN.
Driver	DSN or Driver	In the connector configuration location, Driver should contain the path to the connector binary. In the DSN configuration location, Driver could contain the path to the connector binary, or it could contain the connector entry in the registry.
Locale	DSN	The connection locale.
ServerList	DSN	A list of all servers (IP address and port number) to connect to.
TransactionsSupported	SimbaSettingReader	Overrides the value of DSI_DRIVER_TRANSACTION_CAPABILITY_KNOWN property to specify whether or not the server is connecting to a data store that supports transactions.

Property	Type	Description
Timeout properties		
IdleTimeout	DSN	The time to wait for a response from the server.
LoginTimeout	DSN	The timeout, in seconds, to wait for a response from the server when attempting to log in.
QueryTimeout	DSN	Sets the value of SQL_ATTR_QUERY_TIMEOUT, which is passed to the server.
Logging properties		
LogLevel	SimbaSetting Reader	Controls the granularity of the messages and events that are logged.
LogPath	SimbaSetting Reader	Specifies the directory where the log files are created.
LogFileSize	SimbaSetting Reader	The size of each log file. When the maximum size of the file is reached, a new file is created.
LogFileCount	SimbaSetting Reader	The number of log files to create.
Integrated security (single sign-on) properties		
ServicePrincipalName	DSN	The service principal name for the server with integrated security.
UseIntegratedSecurity	DSN	Indicates whether integrated security (Kerberos) should be used.
Secure socket layer (SSL) properties		

Property	Type	Description
AllowExpiredCert	DSN	Specifies whether the client will accept an expired SSL certificate from the server.
AllowHostMismatch	DSN	Specifies whether the client will accept a hostname from the server that differs from the one in the server's SSL certificate.
AllowSelfSignedCert	DSN	Specifies whether the client accepts SSL certificates from the server which are self-signed.
SSLCACertfile	DSN	Location of the SSL certificate authority certification file.
UseSsl	DSN	Enables SSL encryption for the connection between SimbaClient and SimbaServer.
Auto-Reconnect properties		
AutoReconnect	DSN, Driver, or SimbaSettingReader	Enables or disables auto-reconnect feature.
AutoReconnectAttempts	DSN, Driver, or SimbaSettingReader	The maximum number of attempts the client makes when trying to reconnect to the server.
AutoReconnectCooldown	DSN, Driver, or SimbaSettingReader	The number of seconds the client waits between reconnection attempts.

General configuration properties

ConnectionDialog

The location of the connection dialog file.

Required

Required only if the connection dialog is required.

Allowed values	Allowed characters for file path names. May differ by platform.
Default value	<code>[INSTALL_DIR]\ConnectionDialog.dll</code>
Example	<code>ConnectionDialog=C:\Simba Technologies\InstallDir\ConnectionDialog.dll</code>
Comment	You can use the <code>ConnectionDialog.dll</code> for a staged login when you need to have the client and server have a short conversation before connecting. The Client looks in <code>ODBC.INI</code> first, then <code>ODBCINST.INI</code> , then under Simba Settings .

Description

A brief, human-readable description of the DSN.

Required	No
Allowed values	Any alphanumeric characters.
Default value	<code>Sample SimbaODBCClient DSN</code>
Example	<code>Description=Time system database - read-only.</code>
Comment	This describes the DSN to users who are deciding which DSN to use. Applications will usually display the description of the DSN to help the user choose the correct one.

Driver

The location of the connector file.

Required	No (but recommended)
Allowed values	Allowed characters for file path names. May differ by platform.

Default value	None
Example	<code>Driver= C:\Simba Technologies\InstallDir\SimbaClient.dll</code>
Comment	The <code>Driver</code> keyword points to the location of the ODBC connector itself. This is not always required as the location can be inferred, but we recommend you include it.

Locale

The connection locale.

Required	No
Allowed Values	<p>Values are composed of a 2-letter language code (in lower case), and an optional 2-letter country code (in upper case). If the country code is specified, it must be separated from the language code by a hyphen (-).</p> <p>The language codes conform to the ISO639-2 specification: http://www.loc.gov/standards/iso639-2/php/code_list.php</p> <p>The country codes conform to the ISO3166-1 specification: http://www.iso.org/iso/country_codes/iso-3166-1_decoding_table.htm</p>
Default value	0
Example	<code>Locale=fr-CA</code>
Comment	If this value is set, it overrides the connector-wide locale. For example, the connector-wide locale could be <code>en-US</code> . If the client would prefer <code>fr-CA</code> , it can set the connection locale to <code>fr-CA</code> .

ServerList

A listing of all servers to connect to, including IP address and port number.

Required	Yes
Range	<ul style="list-style-type: none"> IP addresses: Any valid IP address. Ports: 0 - 65535

Default value 127.0.0.1 1543 (loopback with default port)

Example ServerList=127.0.0.1 1543,192.168.0.1 1543

Comment SimbaClient must be able to find SimbaServer on the network. This property enables server discovery. SimbaClient will try to make a network connection to the servers in the order specified until a connection is made.

The format is: <ip> <port>,<ip> <port>,...

TransactionsSupported

Overrides the value of DSI_DRIVER_TRANSACTION_CAPABILITY_KNOWN to specify whether or not the server supports transactions.

Required No

Allowed values

- 0 or FALSE: the server does not support transactions
- 1 or TRUE: the server supports transactions

Default value 0 or FALSE: the server does not support transactions

Example TransactionsSupported=TRUE

Comment This property overrides the server property DSI_DRIVER_TRANSACTION_CAPABILITY_KNOWN, which specifies whether the DSII's `IConnection` implementation knows whether or not transactions are supported, before the connection is established.

For more information on DSI_DRIVER_TRANSACTION_CAPABILITY_KNOWN, see `DSIDriverProperties.h`

Timeout properties

The value of `LoginTimeout` is used by the ODBC client. The values of `IdleTimeout` and `QueryTimeout` are passed to the server, and you must implement any desired timeout behavior in your server's DSII. For more information on timeout behavior, see [How does timeout work?](#) on page 99

IdleTimeout

Sets the value of `SQL_ATTR_CONNECTION_TIMEOUT`, which is passed to the server.

Required	No
Range	0 - UINT_MAX seconds
Default value	0 (no timeout)
Example	<code>IdleTimeout=60000</code>

You must implement any required idle timeout in your server DSN. Typically, this covers any situation that requires a timeout and is not associated with query execution or login. For more information on `SQL_ATTR_CONNECTION_TIMEOUT`, see [https://msdn.microsoft.com/en-us/library/ms713605\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms713605(v=vs.85).aspx).

Comment

Note:

The ODBC client `IdleTimeout` property and the server `IdleTimeout` on page 35 property have the same name, but are different properties.

LoginTimeout

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

Required	No
Range	0 - UINT_MAX seconds
Default value	60
Example	<code>LoginTimeout=10</code>

A value of 0 means no timeout.

The value of this property is used to set the value of SQL_ATTR_LOGIN_TIMEOUT.

Note:

Comment

- 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.
- If additional servers are specified in [ServerList](#), the client attempts to log in to the other servers before timing out.

QueryTimeout

Sets the value of SQL_ATTR_QUERY_TIMEOUT, which is passed to the server.

Required	No
Range	0 - UINT_MAX seconds
Default value	60
Example	QueryTimeout=10
Comment	You must implement any required query timeout in your server DSII. Typically, this means throwing an error if statements such as PREPARE, EXECUTE, or EXECUTEDIRECT take longer than specified by SQL_ATTR_QUERY_TIMEOUT.

Logging properties

For more information on logging, see [Setting Properties to Control Logging](#) on page 81.

LogLevel

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

Required	No
----------	----

Allowed values	See <i>Comment</i> .
Default value	LOG_OFF
Example	LogLevel=LOG_ERROR
Comment	<p>With this keyword, you can control the amount of log output by controlling the kinds of events that are logged.</p> <p>Possible values (case sensitive):</p> <ul style="list-style-type: none"> • 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 where the log file is created.

Required	No
Allowed values	Valid directory path, or unspecified.
Default value	Unspecified
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 SimbaClient.

LogFileSize

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

Required	No
Allowed values	The Simba SDK will accept any positive integer. The maximum size of a file depends on the host machine's specifications.
Default value	20971520 bytes
Example	<code>LogFileSize="30000000"</code>
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 SDK will accept any positive integer. The maximum number of files depends on the host machine's specifications.
Default value	50
Example	<code>LogFileCount=100</code>
Comment	When the maximum number of log files has been created, the oldest file will be deleted and a new one created.

Integrated security (single sign-on) properties

For more information on configuring Kerberos, see [Kerberos Authentication Support](#) on page 85.

ServicePrincipalName

The service principal name for the server with integrated security.

Required	Yes if <code>UseIntegratedSecurity</code> is True.
Allowed values	Any valid service principal name.

Default value	no default
---------------	------------

Example	<code>ServicePrincipalName=<SPN of the server></code>
---------	---

UseIntegratedSecurity

Indicates whether integrated security (single sign-on), should be used when the client establishes a connection to the server. Kerberos authentication is used for integrated security.

Note:

You must configure `UseIntegratedSecurity` on both the `SimbaServer` and the `SimbaClient` in order to use integrated security.

Required	No
----------	----

Allowed values	Disabled, Enabled, Required
----------------	-----------------------------

Default value	Disabled
---------------	----------

Example	<code>UseIntegratedSecurity=Required</code>
---------	---

Comment	If enabled on the client and the server, integrated security will be used during connection.
---------	--

For a summary of the connection types that will be established based on the different client and server settings, see [Configuration Properties for Integrated Security](#) on page 97.

Secure socket layer (SSL) properties

For more information, see [Configuring Secure Sockets Layer \(SSL\)](#) on page 75.

AllowExpiredCert

Specifies whether the client will accept an expired SSL certificate from the server.

Required	No
----------	----

Range	Yes or No (also accepts 1 or 0)
Default value	No
Example	AllowExpiredCert=Yes
Comment	When enabled, the client will accept an expired SSL certificate from the server.

AllowHostMismatch

Specifies whether the client will accept a hostname from the server that differs from the one in the server's SSL certificate.

Required	No
Allowed values	Yes or No (also accepts 1 or 0)
Default value	No
Example	AllowHostMismatch=Yes
Comment	When enabled, the client will accept a different hostname from the server than the one specified in the server's SSL certificate.

AllowSelfSignedCert

Specifies whether the client accepts SSL certificates from the server which are self-signed.

Required	No
Allowed values	Yes or No (also accepts 1 or 0)
Default value	No
Example	AllowSelfSignedCert=Yes

Comment When enabled, the client accepts SSL certificates from the server which are self-signed (as opposed to being signed by a certificate authority).

SSLCACertfile

Location of the SSL certificate authority certification file for SimbaClient to use when encrypting SSL communication with SimbaServer.

Required Yes if `UseSsl` is enabled.

Allowed values Disabled, Enabled, Required

Default value Disabled

Example `SSLCACertFile=C:\Simba Technologies\SimbaEngineSDK\10.1
\Documentation\SSLCertificates\
SampleCACertificate.pem`

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 SimbaClient in order to establish a secure connection. See [Configuration Properties for SSL](#) on page 75.

Required Yes, if `UseSsl` is enabled on the Server.

Allowed values Disabled, Enabled, Required

Default value Disabled

Example `UseSsl=Enabled`

Auto-Reconnect properties

AutoReconnect

Specifies whether the client will attempt to automatically reconnect to the server if the connection is closed or lost.

Required	No
Allowed values	N or Y
Default value	N
Example	<code>AutoReconnect=N</code>
Comment	<p>This feature also requires configuration on the server. For more information, see SimbaServer Configuration Properties on page 33.</p> <p>This property can be set at multiple levels that override each other. Connection String overrides the DSN or SimbaSettingReader, and the DSN overrides the SimbaSettingReader.</p>

AutoReconnectAttempts

Specifies the maximum number of attempts the client makes to reconnect to the server.

Required	No
Allowed values	0 - UINT_MAX
Default value	1
Example	<code>AutoReconnectAttempts=5</code>
Comment	<p>If this property is set to 0 the client makes an unlimited number of reconnection attempts.</p> <p>This property can be set at multiple levels that override each other. Connection String overrides the DSN or SimbaSettingReader, and the DSN overrides the SimbaSettingReader.</p>

AutoReconnectCooldown

Specifies the amount of time, in seconds, the client waits between reconnection attempts.

Required	No
Allowed values	0 - UINT_MAX seconds
Default value	0
Example	<code>AutoReconnectCooldown=5</code>
Comment	<p>Each connection attempt is limited by <code>SQL_ATTR_LOGIN_TIMEOUT</code>, so the total time taken before the auto-reconnect process times out is $(\text{AUTORECONNECTCOOLDOWN} + \text{SQL_ATTR_LOGIN_TIMEOUT}) * \text{AUTORECONNECTATTEMPTS}$.</p> <p>This property can be set at multiple levels that override each other. Connection String overrides the DSN or <code>SimbaSettingReader</code>, and the DSN overrides the <code>SimbaSettingReader</code>.</p>

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.

For information on individual configuration properties, see [SimbaClient for JDBC Configuration Properties](#) on page 66.

Connection URL

The connection URL format for the JDBC client is:

```
jdbc:simba://[HOST];[property]=[value];[property]=[value]
```

For example, in IPv4:

```
jdbc:simba://123.33.2.2:1543,123.6.33.22:1543;  
UID=BartonL;PWD=sneaky;LogLevel=0
```

For example, in IPv6:

```
jdbc:simba://[FE80:0000:0000:0000:0202:B3FF:FE1E:8329]:1543;  
UID=BartonL;PWD=sneaky;LogLevel=0
```

Note:

If you programmatically specify a default port, you do not need to set one in the connection string.

Linking to the connector class

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

- `com.simba.client.core.jdbc4.SCJDBC4Driver` for JDBC 4 connectors
- `com.simba.client.core.jdbc41.SCJDBC41Driver` for JDBC 4.1 connectors
- `com.simba.client.core.jdbc42.SCJDBC4Driver` for JDBC 4.2 connectors

SimbaClient for JDBC Configuration Properties

All configuration properties are configured either programmatically or on the connection string. The following table summarizes the configuration properties. Detailed descriptions are provided in subsequent tables.

Note:

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

Property	Description
UID	User ID.
PWD	User password.
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.
LogPath	Specifies the directory where the log files are created.
Integrated security (single sign-on) properties	
UseIntegratedSecurity	Indicates whether integrated security (Kerberos) should be used.
ServicePrincipalName	The service principal name for the server with integrated security.
Secure socket layer (SSL) properties	
UseSsl	Enable SSL encryption for the connection between SimbaClient and SimbaServer.
TrustedStorePath	The location of the Java keystore.
TrustedStorePassword	The password used to access the trusted key store.
SSLAllowHostMismatch	Specifies whether the client will accept a hostname from the server that differs from the one in the server's SSL certificate.
SSLAllowExpiredCert	Specifies whether the client will accept an expired SSL certificate from the server.
SSLCACertFile	Location of the SSL certificate authority certification file.

General configuration properties

UID

The username to use when accessing the connector.

Required	Yes, if the server requires it.
Allowed values	Any valid string.
Default value	None
Example	UID=jdoe
Comment	The sample Quickstart implementation does not require a username or password in order to establish a connection between the JDB client and the SimbaServer. You can modify the server to require a username and password.

PWD

The password to use when accessing the connector.

Required	Yes if the connector requires it.
Allowed values	Any valid string.
Default value	None
Example	PWD=123Hello

Timeout Properties

For more information on timeout behavior, see [How does timeout work?](#) on page 99

LoginTimeout

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

Required	No
----------	----

Range	0 - UINT_MAX seconds
Default value	60
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.

Note:

Comment

- 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.
- If additional servers are specified in [ServerList](#), the client attempts to log in to the other servers before timing out.

ConnectionTimeout

The timeout, in seconds, to wait for a response from the server after sending a command.

Required	No
Range	0 - UINT_MAX seconds
Default value	0 (no timeout)
Example	ConnectionTimeout=10
Comment	A value of 0 means no timeout.

Logging configuration properties

LogLevel

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

Note:

Log files are not created if this value is set to OFF.

Required	No
Allowed values	See <i>Comment</i> .
Default value	OFF
Example	LogLevel=ERROR
Comment	<p>With this keyword, you can control the amount of log output by controlling the kinds of events that are logged.</p> <p>Possible values (case sensitive):</p> <ul style="list-style-type: none"> • OFF: no logging occurs • FATAL: only log fatal errors • ERROR: log all errors • WARNING: log all errors and warnings • INFO: log all errors, warnings, and informational messages • DEBUG: log method entry and exit points and parameter values for debugging • TRACE: log all method entry points <p>You can also use integers 0 - 6 instead of String, but String is more descriptive.</p>

LogPath

Specifies where the log file is created.

Required	No
Allowed values	Valid directory path, or unspecified.
Default value	Unspecified
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 SimbaClient.

Integrated security (single sign-on) properties

For more information on configuring Kerberos, see [Kerberos Authentication Support](#) on page 85.

ServicePrincipalName

The service principal name for the server with integrated security.

Required	Yes if <code>UseIntegratedSecurity</code> is True.
Allowed values	Any valid service principal name.
Default value	no default
Example	<code>ServicePrincipalName=<SPN of the server></code>

UseIntegratedSecurity

Indicates whether integrated security (single sign-on), should be used when the client establishes a connection to the server. Kerberos authentication is used for integrated security.

Note:

You must configure `UseIntegratedSecurity` on both the `SimbaServer` and the `SimbaClient` in order to use integrated security.

Required	No
Allowed values	Disabled, Enabled, Required
Default value	Disabled
Example	<code>UseIntegratedSecurity=Required</code>

Comment	<p>If enabled on the client and the server, integrated security will be used during connection.</p> <p>For a summary of the connection types that will be established based on the different client and server settings, see Configuration Properties for Integrated Security on page 97.</p>
---------	---

Secure socket layer (SSL) properties

For more information on configuring SSL, see [Configuring Secure Sockets Layer \(SSL\)](#) on page 75.

TrustedStorePath

The location of the Java keystore.

Required	No
Allowed values	Valid absolute or relative directory path to the SSL trusted store.
Default value	no default
Example	<code>TrustedStorePath="C:\JDBCKeyStore"</code>
Comment	<p>Use one of the following options to enable SSL:</p> <ul style="list-style-type: none"> • <code>TrustedStorePath</code> and <code>TrustedStorePassword</code> • Or, <code>SSLCACertFile</code>

TrustedStorePassword

The password used to access the trusted key store.

Required	No
Allowed values	Any valid string.
Default value	none
Example	<code>TrustedStorePassword=SimbaServer1543</code>

Use one of the following options to enable SSL:

Comment

- `TrustedStorePath` and `TrustedStorePassword`
- Or, `SSLCACertFile`

SSLAllowExpiredCert

Specifies whether the client will accept an expired SSL certificate from the server.

Required No

Range Yes or No (also accepts 1 or 0)

Default value No

Example `SSLAllowExpiredCert=Yes`

Comment When enabled, the client will accept an expired SSL certificate from the server.

SSLAllowHostMismatch

Specifies whether the client will accept a hostname from the server that differs from the one in the server's SSL certificate.

Required No

Allowed values Yes or No(also accepts 1 or 0).

Default value No

Example `SSLAllowHostMismatch=Yes`

Comment When enabled, the client will accept a different hostname from the server than the one specified in the server's SSL certificate.

SSLCACertfile

Location of the SSL certificate authority certification file for SimbaClient to use when encrypting SSL communication with SimbaServer.

Required	Yes if UseSsl is enabled.
Allowed values	Disabled, Enabled, Required
Default value	Disabled
Example	SSLCACertFile=C:\Simba Technologies\SimbaEngineSDK\10.1 \Documentation\SSLCertificates\ SampleCACertificate.pem

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 SimbaClient in order to establish a secure connection. See [Configuration Properties for SSL](#) on page 75.

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

Configuring Secure Sockets Layer (SSL)

SimbaClient/Server supports Secure Sockets Layer (SSL) encryption on the connection between SimbaClient and SimbaServer. If SSL is enabled, SimbaClient and 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 SimbaClient and SimbaServer, as explained in [Configuration Properties for SSL](#) on page 75.

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\10.1\Documentation\SSLCertificates` directory. These are Simba self-signed certificates that were created using OpenSSL.

For more information, see [Generating an SSL Certificate with Verisign](#) on page 78 or [Generating a Certificate Authority \(CA\) Certificate for Self-Signing](#) on page 77.

Finally, you must distribute the certificates. See [Distributing SSL Certificates](#) on page 79.

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](#) on page 76.

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.

For more information on setting ODBC client properties for SSL, see [Secure socket layer \(SSL\) properties](#) on page 61. For more information on setting JDBC client properties for SSL, see [Secure socket layer](#)

(SSL) properties on page 72. For more information on setting properties for SSL, see [SSL Configuration Properties](#) on page 42.

UseSSL

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

	Client <code>UseSsl=Disabled</code>	Client <code>UseSsl=Enabled</code>	Client <code>UseSsl=Required</code>
Server <code>UseSsl=Disabled</code>	Connection. No SSL.	Connection. No SSL.	No Connection.
Server <code>UseSsl=Enabled</code>	Connection. No SSL.	Connection with SSL.	Connection with SSL.
Server <code>UseSsl=Required</code>	No Connection.	Connection with SSL.	Connection with SSL.

Example:

- When `SimbaClient` has `UseSsl` set to `Enabled` and `SimbaServer` 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`.

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](#) on page 78.

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 a Certificate Authority \(CA\) Certificate for Self-Signing](#) on page 77.

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 <code>server-key.pem</code> .	The file for the <code>SSLKeyFile</code> configuration keyword for <code>SimbaServer</code> .
Server certificate file, for example <code>server-cert.pem</code> .	The file for the <code>SSLCert-file</code> configuration keyword for <code>SimbaServer</code> .

Certificate Name	Description
Certificate authority file, for example <code>CA-cert.pem</code> .	The file for the <code>SSLCACertFile</code> configuration keyword for <code>SimbaClient</code> .

[Generating an SSL Certificate with Verisign on page 78](#)

[Generating a Certificate Authority \(CA\) Certificate for Self-Signing on page 77](#)

Setting Properties to Control Logging

You can configure the granularity of logging for SimbaClient and SimbaServer. You can also configure the location of log files, and set properties to prevent log files from becoming too large.

Note:

Logging settings are configured on a per-machine basis. These settings apply to every user and DSN on the machine.

What properties can I set to control logging?

For a description of logging properties, see *Logging Configuration Properties* in the following sections:

- [SimbaClient for ODBC Configuration Properties](#) on page 51
- [SimbaServer Configuration Properties](#) on page 33
- [SimbaClient for JDBC Configuration Properties](#) on page 66

Where do I set properties to control logging?

The following table tells you where to find the logging properties.

	Description
SimbaClient for ODBC	See Configuring SimbaClient for ODBC on Windows on page 47 for Windows platforms. See Configuring SimbaClient for ODBC on Linux, Unix, and macOS on page 49 for Linux, Unix, and macOS platforms.
SimbaClient for JDBC	See Configuring SimbaClient for JDBC on page 66.
SimbaServer	See Configuring SimbaServer on Windows on page 29 for Windows platforms. See Configuring SimbaServer on Linux, Unix, and macOS on page 31 for Linux, Unix, and macOS platforms.

Example: Logging Properties for the QuickStart SimbaServer and ODBC Client

The following picture shows a section of the Windows Registry that contains logging configuration properties for the Quickstart server and the Quickstart ODBC client. It also shows the location of the

logging configuration properties for the stand-alone Quickstart connector, for reference.

0

Contact Us

For more information or help using this product, please contact our Technical Support staff. We welcome your questions, comments, and feature requests. You can contact Technical Support via the Magnitude Support Community at www.magnitude.com.

To help us assist you more quickly, please have the following information ready when you contact us:

- The platform and operating system version for the server and client computer(s)
- Your Simba SDK product version.
- The contents of the files listed in the tables below.

SimbaServer on Windows

File	Information
server	Registry key under HKEY_LOCAL_MACHINE\SOFTWARE\Simba\Server
driver	Registry key under HKEY_LOCAL_MACHINE\SOFTWARE\Simba\Driver
odbc	Registry key under HKEY_LOCAL_MACHINE\SOFTWARE\ODBC and/or HKEY_CURRENT_USER\SOFTWARE\ODBC
odbcinst	Registry key under HKEY_LOCAL_MACHINE\SOFTWARE\ODBC

SimbaClient on Windows

File	Information
driver	Registry key under HKEY_LOCAL_MACHINE\SOFTWARE\Simba\Driver
odbc	Registry key under HKEY_CURRENT_USER\SOFTWARE\ODBC
odbcinst	Registry key under HKEY_LOCAL_MACHINE\SOFTWARE\ODBC

SimbaServers and SimbaClients On Linux, Unix, and macOS

File	Information
.odbc.ini	File in SimbaServer user's home directory

File	Information
.profile	File in SimbaServer user's home directory
.simba.ini	File in SimbaServer user's home directory

Kerberos Authentication Support

In addition to plain text authentication, Simba SDK supports authentication using the Kerberos protocol. This allows single sign-on from the client to the server, using the user's current machine credentials (for example the user's Windows password).

The diagram below shows initialization and service requests when Simba SDK uses Kerberos authentication.



Step	Description
1A	The Ticket Granting Ticket (TGT) is requested from the client computer externally from the SimbaClient connector, using the default principal.
1B	The Key Distribution Service returns a TGT and session key.
2A	When the connection to SimbaServer is started, SimbaClient requests a service ticket from the Ticket Granting Service. The request uses the TGT and session key returned for the default principal, and a service request including the Kerberos principal name of the SimbaServer.
2B	The TGS confirms request authenticity, and sends back a service ticket and encrypted service session key.
3A	The SimbaClient requests a service from SimbaServer using the service ticket and service session key.
3B	Authenticates using service session key.
3C	Returns authenticated response and service begins.

When using Microsoft Active Directory to provide Kerberos authentication, Kerberos tickets are requested automatically when the user logs on to a domain. Configure Linux, Unix, and macOS client computers to request Kerberos tickets as needed. For more details, refer to Kerberos documentation.

Example: Configuring Kerberos for C++ Servers

This example shows how to enable Single Sign-on (SSO) with Kerberos between the ODBC client and the Quickstart connector compiled as a server. For simplicity, both the client and the server are deployed on the same Windows machine. Active Directory (AD) Kerberos is used.

To complete this example, you must have administrative access to the Active Directory server.

This example includes the following steps:

- [Step 1: Modify the Server Code on page 86](#)
- [Step 2: Configure the Kerberos SPN on page 87](#)
- [Step 3: Run the Quickstart Server as a Windows Service on page 88](#)
- [Step 4: Update the User for the Quickstart Service on page 88](#)
- [Step 5: Set the Server Configuration Properties on page 88](#)
- [Step 6: Set the ODBC Client Configuration Properties on page 89](#)
- [Step 7: Test SSO with Kerberos Authentication on page 90](#)

Step 1: Modify the Server Code

Modify the Quickstart sample to implement a connection that uses Kerberos credentials, and to set a property specifying that integrated security is used.

1. Implement a Connection class that uses credentials. To do this, open the file `QSEnvironment.h` and declare the function `CreateConnection(ICredentials*)`:

```
#include "ICredentials.h"
....
/** Creates and returns a new IConnection instance with established
credentials.
    @param in_credentials Credentials established by integrated security.
    (NOT OWN)
    @return New IConnection instance. (OWN)
**/
virtual Simba::DSI::IConnection* CreateConnection(ICredentials* in_
credentials);
```

2. Open the file `QSEnvironment.cpp` and add the function declared above:

```
IConnection* QSEnvironment::CreateConnection(ICredentials* in_credentials) { ENTRANCE_LOG(GetLog(), "Quickstart", "QSEnvironment", "CreateConnection(in_credentials)"); return new QSConnection(this); }
```

Note:

You can implement custom security behavior in your connector by extract and use the username, password, and other information from `ICredentials`.

3. Set the driver property value `DSI_DRIVER_SUPPORTS_INTEGRATED_SECURITY` to `DSI_DRIVER_IS_SUPPORTS_KERBEROS`. To do this, open `QSDriver.cpp` and add the following lines of code to `QSDriver::SetDriverPropertyValues()`:

```
#ifdef SERVERTARGET SetProperty(DSI_DRIVER_SUPPORTS_INTEGRATED_SECURITY, AttributeData::MakeNewUInt32AttributeData(DSI_DRIVER_IS_SUPPORTS_KERBEROS)); SetProperty(DSI_DRIVER_SERVICE_PRINCIPAL_NAME, AttributeData::MakeNewWStringAttributeData( new simba_wstring (SimbaSettingReader::ReadSetting("ServicePrincipalName")))); #endif
```

For more information about how this property is used to determine whether a connection is established, see [Configuration Properties for Integrated Security](#) on page 97. For more information about OR-ing other property values with this property, see the file `DataAccessComponents\Include\DSI\DSIDriverProperties.h` in the installation directory of your Simba SDK.

4. Optionally, you can change the service name. In the Quickstart example, the service name is set using the call `SimbaSettingReader::SetServiceName("SimbaQuickstartService")`. In this example, we change the name to `MyQuickstartService`.
5. Build the Quickstart connector as a server. To do this, select `debug_Server` as the solution configuration, then build the solution.

Step 2: Configure the Kerberos SPN

Tip:

This step requires access to the Active Directory server, so you may need to get help from your IT department.

On the Active Directory server, create a Service Principal Name (SPN) associated with the user that you will use to run the Quickstart service. You can choose any name for the service. In this example, we use the following command:

```
setspn -U -A MyQuickstart/Win-DSV05.wd1.sen WD1\KerberosITUser
```

Step 3: Run the Quickstart Server as a Windows Service

Run the Quickstart server as a Windows Service. For more information, see [Configure SimbaServer as a Windows Service](#) on page 20.

If you copied the server to a machine that does not have the Simba SDK installed, you must also copy over the dependencies. See [SimbaServer Required Files](#) on page 22.

Step 4: Update the User for the Quickstart Service

In order for Kerberos authentication to succeed, you must run the Quickstart service under the user account that you specified when creating the SPN.

To update the user account for the Quickstart service:

1. Open Windows Services and double-click the service.
2. Select the **Log On** tab, select **This Account**, and enter the credentials for the user that was configured with the SPN. In this example the SPN is set for the user `WD1\KerberosITUser`, so the user account is set to `wd1\KerberosITUser`.



Step 5: Set the Server Configuration Properties

In the Windows registry editor, under the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Simba\Quickstart\Server`, configure the following values as described in [SimbaServer Configuration Properties](#) on page 33:

- DBF
- DriverLocale
- ErrorMessagePath
- ListenAddress
- ListenPort
- LogLevel
- LogPath

For example:



Step 6: Set the ODBC Client Configuration Properties

If you copy the ODBC client to a machine that does not have the Simba SDK installed, you must also copy over the dependencies. For more information, see [SimbaClient for ODBC Required Files](#) on page 24.

Under the Windows registry key **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources**, configure the client and connector name. For example:

0

Under the registry key **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\<Client Name>**, configure the following keys as described in [SimbaClient for ODBC Configuration Properties](#) on page 51:

- Driver
- LogLevel
- LogPath
- ServerList
- ServicePrincipalName
- UseIntegratedSecurity = Required

0

Under the registry key **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers**, ensure there is an entry for your client that is set to `installed`. For example:

0

Under the registry key **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\<DriverName>**, configure the `Driver` key to point to the location of the connector DLL. For example:

0

Under the registry key **HKEY_LOCAL_MACHINE\SOFTWARE\Simba\SimbaClient\Driver**, configure the following keys:

- ErrorMessagePath
- LogLevel
- LogPath

For example:



Step 7: Test SSO with Kerberos Authentication

Query the server to verify that SSO using Kerberos is configured correctly.

To test the deployment:

1. Log in to the machine as the user that you specified when creating the SPN. In this example, it is `KerberosITUser`.
2. Make sure that the Quickstart server is started as a service, as explained in [Step 4: Update the User for the Quickstart Service](#) on page 88.
3. Navigate to the folder containing the ODBC Test application, by default:
`C:\Program Files (x86)\Microsoft Data Access SDK 2.8\Tools`
4. Navigate to the folder that corresponds to your connector's architecture: **amd64**, **ia64** or **x86**. For example, if you built the 32-bit version of your connector on a 64-bit machine, select the **x86** version.
5. Click one:
 - **odbcte32.exe** to launch the ANSI version
 - Or, **odbct32w.exe** to launch the Unicode version.

Important:

It is important to run the correct version of the ODBC Test tool for ANSI or Unicode and 32-bit or 64-bit.

6. In the ODBC Test tool, click **Conn > Full Connect**.
The Full Connect window opens.
7. In the Full Connect dialog, select your client from the list of data sources, and then click **OK**.
8. In the ODBC Test window, enter `SELECT * from EMP`
9. Click **|** and **|** to output a simple result set. The results are displayed in the window.



10. To validate that the correct `CreateConnection()` method is called, open the server log file. Search for the entry `CreateConnection(in_credentials)` that you added to the server code in step [Step 1: Modify the Server Code](#) on page 86.

Note:

You can use the credential information passed in by Kerberos to enforce custom security in your connector. For more information, see `ICredentials.h` in the `DataAccessComponents\Include` folder of your installation directory.

Example: Configuring Kerberos For Java DSII Connectors

This example shows how to enable Single Sign-on (SSO) with Kerberos between the ODBC client and the JavaQuickstart connector compiled as a server. For simplicity, both the client and the server are deployed on the same Windows machine. Active Directory (AD) Kerberos is used.

The JavaQuickstart connector is an ODBC connector that allows you to write the DSII in Java, then link to the C++ server using a JNI bridge. For an example using the pure-C++ Quickstart connector, see [Example: Configuring Kerberos for C++ Servers](#) on page 86.

To complete this example, you must have administrative access to the Active Directory server.

This example includes the following steps:

- [Step 1: Modify the DSII Code](#) on page 91
- [Step 2: Create the gss.conf and krd5.ini Files](#) on page 93
- [Step 3: Generate the Keytab File](#) on page 94
- [Step 4: Set the Connector Configuration Properties](#) on page 95
- [Step 5: Set the ODBC Client Configuration Properties](#) on page 95
- [Step 6: Test SSO with Kerberos Authentication](#) on page 96

Step 1: Modify the DSII Code

Modify the JavaQuickstart sample to implement a connection that uses Kerberos credentials, and to set a property specifying that integrated security is used.

1. **Implement a constructor for `IConnection` that takes the `ICredentials` parameter. To do this, open the file `QSEnvironment.java` and add the following lines:**

```
import com.simba.support.security.ICredentials;
....
public QSConnection(QSEnvironment environment, ICredentials credentials)
throws Exception { super(environment);
LogUtilities.logFunctionEntrance(getConnectionLog(), environment,
credentials); setDefaultProperties();
```

```
// Read in any configuration and determine if the connector is
// configured to be a server.
// If the file doesn't exist, then the default is the non-server
// configuration.
loadConfiguration();
String setting = m_connConfig.getProperty(TARGET); if (null != setting
&& Boolean.parseBoolean(setting)) { m_isServer = true; } }
```

Note:

You can implement custom security behavior in your connector by extracting and using the username, password, and other information from `ICredentials`.

2. Set the driver property value `DSI_DRIVER_SUPPORTS_INTEGRATED_SECURITY` to `DSI_DRIVER_IS_SUPPORTS_KERBEROS`. To do this, open `QSDriver.java` and add the following lines of code to `QSDriver::setDefaultProperties()`:

```
import
com.simba.dsi.core.utilities.DriverPropertyValues;
setProperty(
    DriverPropertyKey.DSI_DRIVER_SUPPORTS_INTEGRATED_SECURITY,
    new Variant(
        Variant.TYPE_UINT32,
        DriverPropertyValues.DSI_DRIVER_IS_SUPPORTS_KERBEROS
        | DriverPropertyValues.DSI_DRIVER_IS_EXEC_AS_USER));
```

For more information about how this property is used to determine whether a connection is established, see [Configuration Properties for Integrated Security](#) on page 97. For more information about OR-ing other property values with this property, see

`com.simba.dsi.core.utilities.DriverPropertyKey` in the `SimbaEngine Java API Reference` folder of your installation directory.

3. Build the JavaQuickstart connector as a server. To do this:
 - In the `QuickstartJNIDSI` project, select `debug_Server` as the solution configuration, then build the solution.
 - In the `JavaQuickstart` project, build the project as a regular project.

Step 2: Create the gss.conf and krd5.ini Files

To create the gss.conf file:

1. Copy the following lines into a text file, making the following substitutions:
 - For `principal`, add the name of the Kerberos principal. Typically this is a user or group that you want to have SSO access to the JavaQuickstart server. For more information on Kerberos principals, see https://web.mit.edu/kerberos/krb5-1.5/krb5-1.5.4/doc/krb5-user/What-is-a-Kerberos-Principal_003f.html.
 - For `keyTab`, add the path to the keytab file. You create the keytab file in the next step.

```
com.sun.security.jgss.accept {  
  
    com.sun.security.auth.module.Krb5LoginModule required  
    principal="KerberosITUser@WD1.SEN"  
    keyTab="file:C:/SimbaEngine/JavaInternalTestWin64Debug/JIT.keytab"  
    useKeyTab=true  
    storeKey=true  
    doNotPrompt=false  
    debug=true;  
  
};
```

2. Save the file, `gss.conf`, to any folder you choose.

Note:

In this example, we set the following values to facilitate debugging. In production, you may want to set other values.

- `debug=true` enables logging of Kerberos authentication information to standard out.
- `doNotPrompt=false` specifies that Kerberos authentication should prompt for missing information.

To create the krd5.ini file:

1. Copy and paste the following text into a text file, making the replacements described below:
 - For `default_realm`, replace `WD1.SEN` with the name of your default realm.
 - For `kdc`, replace `adserver.wd1.sen` with the full name of your Kerberos domain controller.
 - For `default_domain`, replace `wd1.sen` with the name of your default domain.

```
[libdefaults]
default_realm = WD1.SEN
default_tkt_enctypes = rc4-hmac aes128-cts des3-cbc-sha1 des-cbc-md5
des-cbc-crc
default_tgs_enctypes = rc4-hmac aes128-cts des3-cbc-sha1 des-cbc-md5
des-cbc-crc
permitted_enctypes = rc4-hmac aes128-cts des3-cbc-sha1 des-cbc-md5 des-
cbc-crc
[realms]
WD1.SEN = {
kdc = adserver.wd1.sen
default_domain = wd1.sen
}
```

For more examples of `krb5.conf` files, see https://web.mit.edu/kerberos/krb5-1.12/doc/admin/conf_files/krb5_conf.html#sample-krb5-conf-file.

2. Save the file, `krd5.ini`, to the same folder as in the previous step.

Step 3: Generate the Keytab File

In the same directory that contains the `gss.conf` and `krd5.ini` files, run the following command to create the keytab file:

```
ktpass -out <KeyTab file> -princ <User> -pass <password> -ptype KRB5_NT_
PRINCIPAL -kvno 0
```

where:

- `<KeyTab file>` is the full path to the keytab file you are creating. This must match the keytab file specified when creating the `gss.conf` file
- `<User>` is the principal specified in the `gss.conf` file

For example:

```
ktpass -out C:\SimbaEngine\JavaInternalTestWin64Debug\JIT.keytab -princ
KerberosITUser@WD1.SEN -pass <pass> -ptype KRB5_NT_PRINCIPAL -kvno 0
```

Note:

In this example, `kvno` is set to 0 to specify that Java does not ensure a match with the ActiveDirectory value of `kvno`. In production, you may want to change this to a different value. For more information, see <https://dmdaa.wordpress.com/2010/05/08/how-to-get-needed-kvno-for-keytab-file-created-by-java-ktab-utility>.

Step 4: Set the Connector Configuration Properties

In the Windows registry editor, under the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Simba\Quickstart\Server`, configure the following values as described in [SimbaServer Configuration Properties](#) on page 33:

- DBF
- DriverLocale
- ErrorMessagePath
- ListenAddress
- ListenPort
- LogLevel
- LogPath

For example:



In the same registry key, configure the value **JNIConfig** to contain the values shown below. Note that the pipe character, "|", is used as a delimiter:

```
-Djava.security.auth.login.config=<path to gss.conf>|-  
Djavax.security.auth.useSubjectCredsOnly=false|-  
Djava.security.krb5.conf=<path to krb.ini>
```

For example:

```
-Djava.security.auth.login.config=c:\test\gss.conf|-  
Djavax.security.auth.useSubjectCredsOnly=false|-  
Djava.security.krb5.conf=c:\test\krb.ini
```

Step 5: Set the ODBC Client Configuration Properties

If you copy the ODBC client to a machine that does not have the Simba SDK installed, you must also copy over the dependencies. For more information, see [SimbaClient for ODBC Required Files](#) on page 24.

Under the Windows registry key `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources`, configure the client and connector name. For example:



Under the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\<Client Name>`, configure the following keys as described in [SimbaClient for ODBC Configuration Properties](#) on page 51:

- Driver
- LogLevel
- LogPath
- ServerList
- ServicePrincipalName
- UseIntegratedSecurity = Required



Under the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers`, ensure there is an entry for your client that is set to `installed`. For example:



Under the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\<DriverName>`, configure the `Driver` key to point to the location of the connector DLL. For example:



Under the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Simba\SimbaClient\Driver`, configure the following keys:

- ErrorMessagePath
- LogLevel
- LogPath

For example:



Step 6: Test SSO with Kerberos Authentication

Query the server to verify that SSO using Kerberos is configured correctly.

To test the deployment:

1. Log in to the machine as the user that you specified when creating the keytab file. In this example, it is `KerberosITUser`.
2. Navigate to the folder containing the ODBC Test application, by default:

C:\Program Files (x86)\Microsoft Data Access SDK 2.8\Tools



3. Navigate to the folder that corresponds to your connector's architecture: **amd64**, **ia64** or **x86**. For example, if you built the 32-bit version of your connector on a 64-bit machine, select the **x86** version.
4. Click one:
 - **odbcte32.exe** to launch the ANSI version
 - Or, **odbct32w.exe** to launch the Unicode version.

⚠ Important:

It is important to run the correct version of the ODBC Test tool for ANSI or Unicode and 32-bit or 64-bit.

5. In the ODBC Test tool, click **Conn > Full Connect** to open the Full Connect dialog.
6. In the Full Connect dialog, select your client from the list of data sources, and then click **OK**.
7. In the ODBC Test window, type the following command:

```
SELECT * from EMP
```

8. Click  and  to output a simple result set. The results are displayed in the window.



9. To validate that the correct `CreateConnection()` method is called, open the server log file. Search for the entry `CreateConnection(in_credentials)` that you added to the server code in step [Step 1: Modify the DSII Code on page 91](#).

📘 Note:

You can use the credential information passed in by Kerberos to enforce custom security in your connector. For more information, see `ICredentials.h` in the `DataAccessComponents\Include` folder of your installation directory.

Configuration Properties for Integrated Security

For client-side configuration, the following registry values are used to configure Kerberos authentication:

- **UseIntegratedSecurity**
- **ServicePrincipalName**

For more information on setting these client-side properties, see [SimbaClient for ODBC Configuration Properties](#) on page 51.

To specify that the server use Kerberos, set the property **DSI_DRIVER_SUPPORTS_INTEGRATED_SECURITY** to **DSI_DRIVER_IS_SUPPORTS_KERBEROS**. For more information about OR-ing other property values with this property, see `DSIDriverProperties.h` in `<INSTALL_DIR>\DataAccessComponents\Include\DSI`.

The following table documents the type of connection that is established using different combinations of `UseIntegratedSecurity` on `SimbaClient` and different levels of Kerberos support on `SimbaServer`:

	Client <code>UseIntegratedSecurity</code> is Disabled	Client <code>UseIntegratedSecurity</code> is Enabled	Client <code>UseIntegratedSecurity</code> = Required
Server Kerberos is Not Supported	Connection. No Kerberos.	Connection. No Kerberos.	No Connection.
Server Kerberos is Supported	Connection. No Kerberos.	Connection only if Kerberos authentication succeeds.	Connection only if Kerberos authentication succeeds.
Server Kerberos is Supported with Fallback	Connection. No Kerberos.	Connection only if Kerberos authentication or basic authentication succeeds.	Connection only if Kerberos authentication succeeds.
Server Kerberos is Required	No Connection.	Connection only if Kerberos authentication succeeds.	Connection only if Kerberos authentication succeeds.

For example, if `SimbaClient` has `UseIntegratedSecurity` set to 1 and `SimbaServer` is configured to require Kerberos, a connection will be established only if Kerberos authentication succeeds.

Frequently Asked Questions

How does timeout work?

Timeout properties control the behavior of different types of timeouts on the connection between the client and the server. Typically, the timeout value is set on the client but implemented on the server. The only timeout that is implemented by the client is the login timeout.

You must provide implementation for some types of timeouts in your server DSII, while other timeouts are handled automatically by the Simba SDK and are not configurable.

The following table summarizes the timeout configuration properties:

Timeout Property	Description
IdleTimeout (on the client)	This property specifies the timeout defined by <code>SQL_ATTR_CONNECTION_TIMEOUT</code> , which covers any situation that requires a timeout and is not associated with query execution or login. The value is set on the client, and you must implement the timeout behavior in your server DSII.
IdleTimeout (on the server)	The duration in seconds that a connection can remain idle, with no communication from a client, before <code>SimbaServer</code> disconnects it. This timeout behavior is implemented automatically in the Simba SDK.
LoginTimeout	This property specifies the timeout defined by <code>SQL_ATTR_LOGIN_TIMEOUT</code> , which is the time to wait for a response from the server after a login request is made by the client. The behavior is implemented in the ODBC and JDBC client.
QueryTimeout	This property controls the timeout defined by <code>SQL_ATTR_QUERY_TIMEOUT</code> , which is the timeout during query and command execution. The value is set on the client, and you must implement the timeout behaviour in your server DSII.

For more information on setting the timeout properties, see the following sections:

- For information on setting timeout properties in the ODBC client, see [Timeout properties](#) on page 56.
- For information on setting timeout properties in the JDBC client, see [Timeout Properties](#) on page 68.
- For information on setting timeout properties in the server, see [IdleTimeout](#) on page 35.

Keepalive messages

The ODBC and JDBC client sends keepalive messages to the server once a minute to ensure that the connection is valid. If the connection is no longer valid, the client terminates it. The frequency of keepalive messages is not configurable.

How can I build the ODBC or JDBC client?

The ODBC and JDBC clients are shipped by Simba Technologies Inc., so you do not need to compile them. See [Example: ODBC Client/Server Deployment](#) on page 10 for step-by-step instructions on deploying the QuickStart sample connector in a client/server deployment.

Third-Party Trademarks

Simba, the Simba logo, Simba SDK, and Simba Technologies are registered trademarks of Simba Technologies Inc. in Canada, United States and/or other countries. All other trademarks and/or servicemarks are the property of their respective owners.

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

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

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

Microsoft SQL Server, SQL Server, Microsoft, MSDN, Windows, Windows Azure, 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.

Solaris is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

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.

All other trademarks are trademarks of their respective owners.