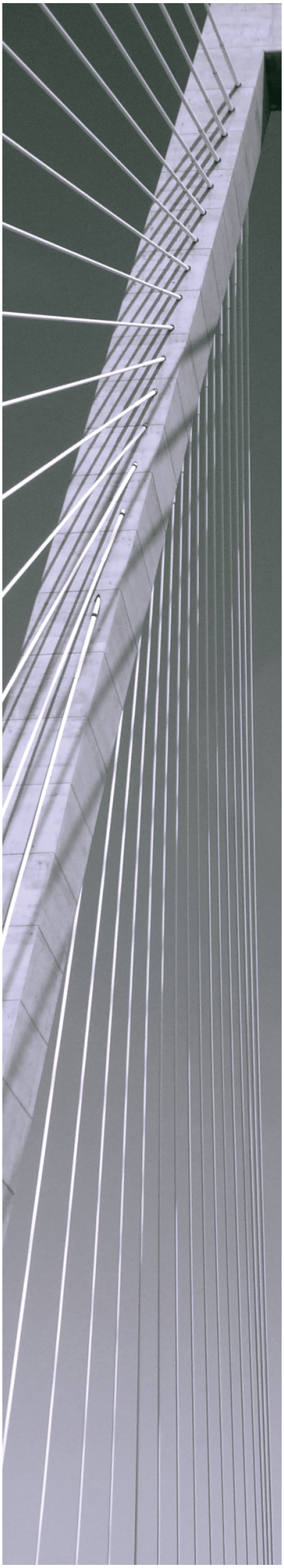

Simba Hive JDBC Driver with SQL Connector

Installation and Configuration Guide

Simba Technologies Inc.

Version 2.6.7

May 10, 2019

Copyright © 2019 Simba Technologies Inc. All Rights Reserved.

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this publication, or the software it describes, may be reproduced, transmitted, transcribed, stored in a retrieval system, decompiled, disassembled, reverse-engineered, or translated into any language in any form by any means for any purpose without the express written permission of Simba Technologies Inc.

Trademarks

Simba, the Simba logo, SimbaEngine, 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.

Contact Us

Simba Technologies Inc.
938 West 8th Avenue
Vancouver, BC Canada
V5Z 1E5

Tel: +1 (604) 633-0008

Fax: +1 (604) 633-0004

www.simba.com

About This Guide

Purpose

The *Simba Hive JDBC Driver with SQL Connector Installation and Configuration Guide* explains how to install and configure the Simba Hive JDBC Driver with SQL Connector on all supported platforms. The guide also provides details related to features of the driver.

Audience

The guide is intended for end users of the Simba Hive JDBC Driver.

Knowledge Prerequisites

To use the Simba Hive JDBC Driver, the following knowledge is helpful:

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

Document Conventions

Italics are used when referring to book and document titles.

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

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

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

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

Table of Contents

About the Simba Hive JDBC Driver 6

System Requirements 7

Simba Hive JDBC Driver Files 8

Installing and Using the Simba Hive JDBC Driver 9

- Referencing the JDBC Driver Libraries 9
- Registering the Driver Class 10
- Building the Connection URL 11

Configuring Authentication 13

- Using No Authentication 13
- Using Kerberos 14
- Using User Name 15
- Using User Name And Password (LDAP) 15
- Using a Hadoop Delegation Token 16
- Authentication Mechanisms 19
- Configuring Kerberos Authentication for Windows 21
- Kerberos Encryption Strength and the JCE Policy Files Extension 26

Configuring SSL 29

Configuring Server-Side Properties 31

Configuring Logging 32

Features 34

- SQL Query versus HiveQL Query 34
- Data Types 34
- Catalog and Schema Support 35
- Write-back 36
- IHadoopStatement 36
- IHadoopConnection 40
- Security and Authentication 43

Driver Configuration Options 45

- AllowSelfSignedCerts 45
- AsyncExecPollInterval 46
- AuthMech 46

CAIssuedCertsMismatch 47
CatalogSchemaSwitch 47
DecimalColumnScale 47
DefaultStringColumnLength 48
DelegationToken 48
DelegationUID 48
httpPath 49
KrbAuthType 49
KrbHostFQDN 50
KrbRealm 51
KrbServiceName 51
LogLevel 51
LogPath 52
PreparedMetaLimitZero 53
PWD 53
RowsFetchedPerBlock 53
SocketTimeout 54
SSL 54
SSLKeyStore 55
SSLKeyStorePwd 55
SSLTrustStore 56
SSLTrustStorePwd 56
transportMode 57
UID 57
UseNativeQuery 58
zk 58

Contact Us 60

Third-Party Trademarks 61

About the Simba Hive JDBC Driver

The Simba Hive JDBC Driver is used for direct SQL and HiveQL access to Apache Hadoop / Hive distributions, enabling Business Intelligence (BI), analytics, and reporting on Hadoop / Hive-based data. The driver efficiently transforms an application's SQL query into the equivalent form in HiveQL, which is a subset of SQL-92. If an application is Hive-aware, then the driver is configurable to pass the query through to the database for processing. The driver interrogates Hive to obtain schema information to present to a SQL-based application. Queries, including joins, are translated from SQL to HiveQL. For more information about the differences between HiveQL and SQL, see Features on page 34.

The Simba Hive JDBC Driver complies with the JDBC 4.0 and 4.1 data standards. JDBC is one of the most established and widely supported APIs for connecting to and working with databases. At the heart of the technology is the JDBC driver, which connects an application to the database. For more information about JDBC, see *Data Access Standards* on the Simba Technologies website: https://www.simba.com/resources/data-access-standards-glossary.

This guide is suitable for users who want to access data residing within Hive from their desktop environment. Application developers might also find the information helpful. Refer to your application for details on connecting via JDBC.

System Requirements

Each machine where you use the Simba Hive JDBC Driver must have Java Runtime Environment (JRE) 7.0 or 8.0 installed.

The driver supports Apache Hive versions 0.11 through 3.1.

Simba Hive JDBC Driver Files

The Simba Hive JDBC Driver is delivered in the following ZIP archives, where *[Version]* is the version number of the driver:

- `HiveJDBC4_[Version].zip`
- `HiveJDBC41_[Version].zip`

The archive contains the driver supporting the JDBC API version indicated in the archive name, as well as release notes and third-party license information. In addition, the required third-party libraries and dependencies are packaged and shared in the driver JAR file in the archive.

Installing and Using the Simba Hive JDBC Driver

To install the Simba Hive JDBC Driver on your machine, extract the files from the appropriate ZIP archive to the directory of your choice.

> **! Important:**
>
> If you received a license file through email, then you must copy the file into the same directory as the driver JAR file before you can use the Simba Hive JDBC Driver.

To access a Hive data store using the Simba Hive JDBC Driver, you need to configure the following:

- The list of driver library files (see Referencing the JDBC Driver Libraries on page 9)
- The `Driver` or `DataSource` class (see Registering the Driver Class on page 10)
- The connection URL for the driver (see Building the Connection URL on page 11)

Referencing the JDBC Driver Libraries

Before you use the Simba Hive JDBC Driver, the JDBC application or Java code that you are using to connect to your data must be able to access the driver JAR files. In the application or code, specify all the JAR files that you extracted from the ZIP archive.

Using the Driver in a JDBC Application

Most JDBC applications provide a set of configuration options for adding a list of driver library files. Use the provided options to include all the JAR files from the ZIP archive as part of the driver configuration in the application. For more information, see the documentation for your JDBC application.

Using the Driver in Java Code

You must include all the driver library files in the class path. This is the path that the Java Runtime Environment searches for classes and other resource files. For more information, see "Setting the Class Path" in the appropriate Java SE Documentation.

For Java SE 7:

- For Windows: http://docs.oracle.com/javase/7/docs/technotes/tools/windows/classpath.html
- For Linux and Solaris: http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/classpath.html

For Java SE 8:

- For Windows: http://docs.oracle.com/javase/8/docs/technotes/tools/windows/classpath.html
- For Linux and Solaris: http://docs.oracle.com/javase/8/docs/technotes/tools/solaris/classpath.html

Registering the Driver Class

Before connecting to your data, you must register the appropriate class for your application.

The following is a list of the classes used to connect the Simba Hive JDBC Driver to Hive data stores. The `Driver` classes extend `java.sql.Driver`, and the `DataSource` classes extend `javax.sql.DataSource` and `javax.sql.ConnectionPoolDataSource`.

To support JDBC 4.0, classes with the following fully-qualified class names (FQCNs) are available:

- `com.simba.hive.jdbc4.HS1Driver`
- `com.simba.hive.jdbc4.HS2Driver`
- `com.simba.hive.jdbc4.HS1DataSource`
- `com.simba.hive.jdbc4.HS2DataSource`

To support JDBC 4.1, classes with the following FQCNs are available:

- `com.simba.hive.jdbc41.HS1Driver`
- `com.simba.hive.jdbc41.HS2Driver`
- `com.simba.hive.jdbc41.HS1DataSource`
- `com.simba.hive.jdbc41.HS2DataSource`

The following sample code shows how to use the `DriverManager` to establish a connection for JDBC 4.0:

> **Note:**
>
> In these examples, the line `Class.forName(DRIVER_CLASS);` is only required for JDBC 4.0.

```
private static Connection connectViaDM() throws Exception
{
   Connection connection = null;
   Class.forName(DRIVER_CLASS);
   connection = DriverManager.getConnection(CONNECTION_URL);
   return connection;
}
```

The following sample code shows how to use the `DataSource` class to establish a connection:

```
private static Connection connectViaDS() throws Exception
{
   Connection connection = null;
   Class.forName(DRIVER_CLASS);
   DataSource ds = new com.simba.hive.jdbc41.HS1DataSource
   ();
   ds.setURL(CONNECTION_URL);
   connection = ds.getConnection();
   return connection;
}
```

Building the Connection URL

Use the connection URL to supply connection information to the data source that you are accessing. The following is the format of the connection URL for the Simba Hive JDBC Driver, where *[Subprotocol]* is **hive** if you are connecting to a Hive Server 1 instance or **hive2** if you are connecting to a Hive Server 2 instance, *[Host]* is the DNS or IP address of the Hive server, and *[Port]* is the number of the TCP port that the server uses to listen for client requests:

```
jdbc:[Subprotocol]://[Host]:[Port]
```

Note:

By default, Hive uses port 10000.

By default, the driver uses the schema named **default** and authenticates the connection using the user name **anonymous**.

You can specify optional settings such as the number of the schema to use or any of the connection properties supported by the driver. For a list of the properties available in the driver, see Driver Configuration Options on page 45.

Note:

If you specify a property that is not supported by the driver, then the driver attempts to apply the property as a Hive server-side property for the client session. For more information, see Configuring Server-Side Properties on page 31.

The following is the format of a connection URL that specifies some optional settings:

```
jdbc:[Subprotocol]://[Host]:[Port]/[Schema];[Property1]=
[Value];[Property2]=[Value];...
```

For example, to connect to port 11000 on a Hive Server 2 instance installed on the local machine, use a schema named default2, and authenticate the connection using a user name and password, you would use the following connection URL:

```
jdbc:hive2://localhost:11000/default2;AuthMech=3;
UID=simba;PWD=simba
```

Important:

- Properties are case-sensitive.
- Do not duplicate properties in the connection URL.

Note:

- If you specify a schema in the connection URL, you can still issue queries on other schemas by explicitly specifying the schema in the query. To inspect your databases and determine the appropriate schema to use, run the `show databases` command at the Hive command prompt.
- If you set the `transportMode` property to `http`, then the port number specified in the connection URL corresponds to the HTTP port rather than the TCP port. By default, Hive servers use 10001 as the HTTP port number.

Configuring Authentication

The Simba Hive JDBC Driver supports the following authentication mechanisms:

- No Authentication
- Kerberos
- User Name
- User Name And Password
- Hadoop Delegation Token

You configure the authentication mechanism that the driver uses to connect to Hive by specifying the relevant properties in the connection URL.

For information about selecting an appropriate authentication mechanism when using the Simba Hive JDBC Driver, see Authentication Mechanisms on page 19.

For information about the properties you can use in the connection URL, see Driver Configuration Options on page 45.

> **Note:**
>
> In addition to authentication, you can configure the driver to connect over SSL. For more information, see Configuring SSL on page 29.

Using No Authentication

> **Note:**
>
> When connecting to a Hive server of type Hive Server 1, you must use No Authentication.

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

To configure a connection without authentication:

1. Set the `AuthMech` property to `0`.
2. Set the `transportMode` property to `binary`.

For example:

```
jdbc:hive
2://localhost:10000;AuthMech=0;transportMode=binary;
```

Using Kerberos

Kerberos must be installed and configured before you can use this authentication mechanism. For information about configuring and operating Kerberos on Windows, see Configuring Kerberos Authentication for Windows on page 21. For other operating systems, see the MIT Kerberos documentation: http://web.mit.edu/kerberos/krb5-latest/doc/.

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

Note:

- This authentication mechanism is available only for Hive Server 2.
- For this authentication mechanism, SASL and HTTP Thrift transport protocols are supported. If the `transportMode` property is not set, the driver defaults SASL. If the hive.server2.transport.mode property has been set to HTTP on the server side, set the `transportMode` property to `http`.

To configure default Kerberos authentication:

1. Set the `AuthMech` property to `1`.
2. To use the default realm defined in your Kerberos setup, do not set the `KrbRealm` property.

   If your Kerberos setup does not define a default realm or if the realm of your Hive server is not the default, then set the `KrbRealm` property to the realm of the Hive server.
3. Set the `KrbHostFQDN` property to the fully qualified domain name of the Hive server host.
4. Optionally, specify how the driver obtains the Kerberos Subject by setting the `KrbAuthType` property as follows:
   - To configure the driver to automatically detect which method to use for obtaining the Subject, set the `KrbAuthType` property to `0`. Alternatively, do not set the `KrbAuthType` property.
   - Or, to create a LoginContext from a JAAS configuration and then use the Subject associated with it, set the `KrbAuthType` property to `1`.
   - Or, to create a LoginContext from a Kerberos ticket cache and then use the Subject associated with it, set the `KrbAuthType` property to `2`.

   For more detailed information about how the driver obtains Kerberos Subjects based on these settings, see KrbAuthType on page 49.

For example, the following connection URL connects to a Hive server with Kerberos enabled, but without SSL enabled:

```
jdbc:hive2://node1.example.com:10000;AuthMech=1;
KrbRealm=EXAMPLE.COM;KrbHostFQDN=hs2node1.example.com;
KrbServiceName=hive;KrbAuthType=2
```

In this example, Kerberos is enabled for JDBC connections, the Kerberos service principal name is hive/node1.example.com@EXAMPLE.COM, the host name for the data source is node1.example.com, and the server is listening on port 10000 for JDBC connections.

Using User Name

This authentication mechanism requires a user name but does not require a password. The user name labels the session, facilitating database tracking.

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

> **Note:**
>
> This authentication mechanism is available only for Hive Server 2. Most default configurations of Hive Server 2 require User Name authentication.

To configure User Name authentication:

1. Set the `AuthMech` property to `2`.
2. Set the `transportMode` property to `sasl`.
3. Set the `UID` property to an appropriate user name for accessing the Hive server.

For example:

```
jdbc:hive2://node1.example.com:10000;AuthMech=2;
transportMode=sasl;UID=hs2
```

Using User Name And Password (LDAP)

This authentication mechanism requires a user name and a password. It is most commonly used with LDAP authentication.

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

Note:

This authentication mechanism is available only for Hive Server 2.

To configure User Name And Password authentication:

1. Set the `AuthMech` property to `3`.
2. Set the `transportMode` property to the transport protocol that you want to use in the Thrift layer.
3. If you set the `transportMode` property to `http`, then set the `httpPath` property to the partial URL corresponding to the Hive server. Otherwise, do not set the `httpPath` property.
4. Set the `UID` property to an appropriate user name for accessing the Hive server.
5. Set the `PWD` property to the password corresponding to the user name you provided.

For example, the following connection URL connects to a Hive server with LDAP authentication enabled, but without SSL or SASL enabled:

```
jdbc:hive2://node1.example.com:10001;AuthMech=3;
transportMode=http;httpPath=cliservice;UID=hs2;PWD=simba;
```

In this example, user name and password (LDAP) authentication is enabled for JDBC connections, the LDAP user name is hs2, the password is simba, and the server is listening on port 10001 for JDBC connections.

Using a Hadoop Delegation Token

This authentication mechanism requires a Hadoop delegation token. This token must be provided to the driver in the form of a Base64 URL-safe encoded string. It can be obtained from the driver using the `getDelegationToken()` function, or by utilizing the Hadoop distribution `.jar` files. For a code sample that demonstrates how to retrieve the token using the `getDelegationToken()` function, see Code Samples: Retrieving a Hadoop Delegation Token on page 17.

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

Note:

- This authentication mechanism is available only for Hive Server 2.
- This authentication mechanism requires that Kerberos be configured on the server.

To configure Hadoop delegation token authentication:

1. Make sure Kerberos is configured on the server.
2. Set the `AuthMech` property to `6`.
3. Set the `delegationToken` property to an appropriately encoded Hadoop delegation token.

For example:

```
jdbc:hive
2
://node1.example.com:
10000;AuthMech=6;delegationToken=kP9PcyQ7prK2LwUMZMpFQ4R+5VE
```

Code Samples: Retrieving a Hadoop Delegation Token

If you are using a Hadoop delegation token for authentication, the token must be provided to the driver in the form of a Base64 URL-safe encoded string. This token can be obtained from the driver using the `getDelegationToken()` function, or by utilizing the Hadoop distribution `.jar` files.

The code samples below demonstrate the use of the `getDelegationToken()` function. For more information about this function, see IHadoopConnection on page 40.

The sample below shows how to obtain the token string with the driver using a Kerberos connection:

```
import
com.simba.hiveserver2.hivecommon.core.IHadoopConnection;
public class TestDriverGetDelegationTokenClass
{
   public static void main(String[] args) throws
   SQLException
   {
      // Create the connection object with Kerberos
      authentication.
      Connection kerbConnection =
      DriverManager.getConnection(
         "jdbc:hive2://localhost:10000;AuthMech=1;KrbRealm=Y
         ourRealm;KrbHostFQDN=sample.com;KrbServiceName=hiv
         e;");
      // Unwrap the java.sql.Connection object to an
      implementation of IHadoopConnection so the
```

```
      // methods for delegation token can be called.
      String delegationToken = kerbConnection.unwrap
      (IHadoopConnection.class).getDelegationToken("owner_
      name", "renewer_name");

      // The token can then be used to connect with the
      driver.
      String tokenConnectionString =
      "jdbc:hive2://localhost:10000;AuthMech=6;DelegationTok
      en=" + delegationToken;
      Connection tokenConnection =
      DriverManager.getConnection(tokenConnectionString);
   }
}
```

The sample below demonstrates how to obtain the encoded string form of the token if the delegation is saved to the UserGroupInformation. This sample requires the `hadoop-shims-common-`*`[hadoop version]`*`.jar`, `hadoop-common-`*`[hadoop version]`*`.jar`, and all their dependencies.

```
import org.apache.hadoop.hive.shims.Utils;
import org.apache.hive.service.auth.HiveAuthFactory;
public class TestHadoopDelegationTokenClass
{
   public static void main(String[] args) throws
   SQLException
   {
      // Obtain the delegationToken stored in the current
      UserGroupInformation.
      String delegationToken = Utils.getTokenStrForm
      (HiveAuthFactory.HS2_CLIENT_TOKEN);

      // The token can then be used to connect with the
      driver.
      String tokenConnectionString =
      "jdbc:hive2://localhost:10000;AuthMech=6;DelegationTok
      en=" + delegationToken;
      Connection tokenConnection =
      DriverManager.getConnection(tokenConnectionString);
   }
}
```

Authentication Mechanisms

To connect to a Hive server, you must configure the Simba Hive JDBC Driver to use the authentication mechanism that matches the access requirements of the server and provides the necessary credentials. To determine the authentication settings that your Hive server requires, check the server configuration and then refer to the corresponding section below.

Hive Server 1

Hive Server 1 does not support authentication. You must configure the driver to use No Authentication (see Using No Authentication on page 13).

Hive Server 2

Hive Server 2 supports the following authentication mechanisms:

- No Authentication (see Using No Authentication on page 13)
- Kerberos (see Using Kerberos on page 14)
- User Name (see Using User Name on page 15)
- User Name And Password (see Using User Name And Password (LDAP) on page 15)
- Hadoop Delegation Token (see Using a Hadoop Delegation Token on page 16)

Most default configurations of Hive Server 2 require User Name authentication. If you are unable to connect to your Hive server using User Name authentication, then verify the authentication mechanism configured for your Hive server by examining the `hive-site.xml` file. Examine the following properties to determine which authentication mechanism your server is set to use:

- `hive.server2.authentication`: This property sets the authentication mode for Hive Server 2. The following values are available:
  - `NONE` enables plain SASL transport. This is the default value.
  - `NOSASL` disables the Simple Authentication and Security Layer (SASL).
  - `KERBEROS` enables Kerberos authentication and delegation token authentication.
  - `PLAINSASL` enables user name and password authentication using a cleartext password mechanism.
  - `LDAP` enables user name and password authentication using the Lightweight Directory Access Protocol (LDAP).
- `hive.server2.enable.doAs`: If this property is set to the default value of `TRUE`, then Hive processes queries as the user who submitted the query. If this

property is set to `FALSE`, then queries are run as the user that runs the `hiveserver2` process.

The following table lists the authentication mechanisms to configure for the driver based on the settings in the `hive-site.xml` file.

hive.server2.authentication	hive.server2.enable.doAs	Driver Authentication Mechanism
`NOSASL`	`FALSE`	No Authentication
`KERBEROS`	`TRUE` or `FALSE`	Kerberos
`KERBEROS`	`TRUE`	Delegation Token
`NONE`	`TRUE` or `FALSE`	User Name
`PLAINSASL` or `LDAP`	`TRUE` or `FALSE`	User Name And Password

Note:

It is an error to set `hive.server2.authentication` to `NOSASL` and `hive.server2.enable.doAs` to `true`. This configuration will not prevent the service from starting up, but results in an unusable service.

For more information about authentication mechanisms, refer to the documentation for your Hadoop / Hive distribution. See also "Running Hadoop in Secure Mode" in the Apache Hadoop documentation: http://hadoop.apache.org/docs/r0.23.7/hadoop-project-dist/hadoop-common/ClusterSetup.html#Running_Hadoop_in_Secure_Mode.

Using No Authentication

When `hive.server2.authentication` is set to `NOSASL`, you must configure your connection to use No Authentication.

Using Kerberos

When connecting to a Hive Server 2 instance and `hive.server2.authentication` is set to `KERBEROS`, you must configure your connection to use Kerberos or Delegation Token authentication.

Using User Name

When connecting to a Hive Server 2 instance and `hive.server2.authentication` is set to `NONE`, you must configure your connection to use User Name authentication. Validation of the credentials that you include depends on `hive.server2.enable.doAs`:

- If `hive.server2.enable.doAs` is set to `TRUE`, then the server attempts to map the user name provided by the driver from the driver configuration to an existing operating system user on the host running Hive Server 2. If this user name does not exist in the operating system, then the user group lookup fails and existing HDFS permissions are used. For example, if the current user group is allowed to read and write to the location in HDFS, then read and write queries are allowed.
- If `hive.server2.enable.doAs` is set to `FALSE`, then the user name in the driver configuration is ignored.

If no user name is specified in the driver configuration, then the driver defaults to using **hive** as the user name.

Using User Name And Password

When connecting to a Hive Server 2 instance and the server is configured to use the SASL-PLAIN authentication mechanism with a user name and a password, you must configure your connection to use User Name And Password authentication.

Configuring Kerberos Authentication for Windows

You can configure your Kerberos setup so that you use the MIT Kerberos Ticket Manager to get the Ticket Granting Ticket (TGT), or configure the setup so that you can use the driver to get the ticket directly from the Key Distribution Center (KDC). Also, if a client application obtains a Subject with a TGT, it is possible to use that Subject to authenticate the connection.

Downloading and Installing MIT Kerberos for Windows

To download and install MIT Kerberos for Windows 4.0.1:

1. Download the appropriate Kerberos installer:
    - For a 64-bit machine, use the following download link from the MIT Kerberos website: http://web.mit.edu/kerberos/dist/kfw/4.0/kfw-4.0.1-amd64.msi.

- For a 32-bit machine, use the following download link from the MIT Kerberos website: http://web.mit.edu/kerberos/dist/kfw/4.0/kfw-4.0.1-i386.msi.

> **Note:**
>
> The 64-bit installer includes both 32-bit and 64-bit libraries. The 32-bit installer includes 32-bit libraries only.

2. To run the installer, double-click the `.msi` file that you downloaded.
3. Follow the instructions in the installer to complete the installation process.
4. When the installation completes, click **Finish**.

Using the MIT Kerberos Ticket Manager to Get Tickets

Setting the KRB5CCNAME Environment Variable

You must set the KRB5CCNAME environment variable to your credential cache file.

To set the KRB5CCNAME environment variable:

1. Click **Start**, then right-click **Computer**, and then click **Properties**.
2. Click **Advanced System Settings**.
3. In the System Properties dialog box, on the **Advanced** tab, click **Environment Variables**.
4. In the Environment Variables dialog box, under the System Variables list, click **New**.
5. In the **New System Variable** dialog box, in the Variable Name field, type **KRB5CCNAME**.
6. In the **Variable Value** field, type the path for your credential cache file. For example, type `C:\KerberosTickets.txt`.
7. Click **OK** to save the new variable.
8. Make sure that the variable appears in the System Variables list.
9. Click **OK** to close the Environment Variables dialog box, and then click **OK** to close the System Properties dialog box.
10. Restart your machine.

Getting a Kerberos Ticket

To get a Kerberos ticket:

1. Click **Start**, then click **All Programs**, and then click the **Kerberos for Windows (64-bit)** or **Kerberos for Windows (32-bit)** program group.
2. Click **MIT Kerberos Ticket Manager**.

3. In the MIT Kerberos Ticket Manager, click **Get Ticket**.
4. In the Get Ticket dialog box, type your principal name and password, and then click **OK**.

If the authentication succeeds, then your ticket information appears in the MIT Kerberos Ticket Manager.

Authenticating to the Hive Server

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

To authenticate to the Hive server:

- Use a connection URL that has the following properties defined:
  - `AuthMech`
  - `KrbHostFQDN`
  - `KrbRealm`
  - `KrbServiceName`

For detailed information about these properties, see Driver Configuration Options on page 45

Using the Driver to Get Tickets

Deleting the KRB5CCNAME Environment Variable

To enable the driver to get Ticket Granting Tickets (TGTs) directly, make sure that the KRB5CCNAME environment variable has not been set.

To delete the KRB5CCNAME environment variable:

1. Click the **Start** button , then right-click **Computer**, and then click **Properties**.
2. Click **Advanced System Settings**.
3. In the System Properties dialog box, click the **Advanced** tab and then click **Environment Variables**.
4. In the Environment Variables dialog box, check if the KRB5CCNAME variable appears in the System variables list. If the variable appears in the list, then select the variable and click **Delete**.
5. Click **OK** to close the Environment Variables dialog box, and then click **OK** to close the System Properties dialog box.

Setting Up the Kerberos Configuration File

To set up the Kerberos configuration file:

1. Create a standard `krb5.ini` file and place it in the `C:\Windows` directory.
2. Make sure that the KDC and Admin server specified in the `krb5.ini` file can be resolved from your terminal. If necessary, modify `C:\Windows\System32\drivers\etc\hosts`.

Setting Up the JAAS Login Configuration File

To set up the JAAS login configuration file:

1. Create a JAAS login configuration file that specifies a keytab file and `doNotPrompt=true`.

   For example:

   ```
   Client {
   com.sun.security.auth.module.Krb5LoginModule required
   useKeyTab=true
   keyTab="PathToTheKeyTab"
   principal="simba@SIMBA"
   doNotPrompt=true;
   };
   ```

2. Set the `java.security.auth.login.config` system property to the location of the JAAS file.

   For example: `C:\KerberosLoginConfig.ini`.

Authenticating to the Hive Server

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

To authenticate to the Hive server:

- Use a connection URL that has the following properties defined:
  - `AuthMech`
  - `KrbHostFQDN`
  - `KrbRealm`
  - `KrbServiceName`

  For detailed information about these properties, see Driver Configuration Options on page 45.

Using an Existing Subject to Authenticate the Connection

If the client application obtains a Subject with a TGT, then that Subject can be used to authenticate the connection to the server.

To use an existing Subject to authenticate the connection:

1. Create a PrivilegedAction for establishing the connection to the database.

   For example:

```
// Contains logic to be executed as a privileged action
public class AuthenticateDriverAction
implements PrivilegedAction<Void>
{
// The connection, which is established as a
PrivilegedAction
Connection con;

// Define a string as the connection URL
static String ConnectionURL =
"jdbc:hive2://192.168.1.1:10000";

/**
* Logic executed in this method will have access to the
* Subject that is used to "doAs". The driver will get
* the Subject and use it for establishing a connection
* with the server.
*/
@Override
public Void run()
{
try
{
// Establish a connection using the connection URL
con = DriverManager.getConnection(ConnectionURL);
}
catch (SQLException e)
{
// Handle errors that are encountered during
// interaction with the data store
e.printStackTrace();
}
catch (Exception e)
```

```
{
// Handle other errors
e.printStackTrace();
}
return null;
}
}
```

2. Run the PrivilegedAction using the existing Subject, and then use the connection.

   For example:

```
// Create the action
AuthenticateDriverAction authenticateAction = new
AuthenticateDriverAction();
// Establish the connection using the Subject for
// authentication.
Subject.doAs(loginConfig.getSubject(),
authenticateAction);
// Use the established connection.
authenticateAction.con;
```

Kerberos Encryption Strength and the JCE Policy Files Extension

If the encryption being used in your Kerberos environment is too strong, you might encounter the error message "Unable to connect to server: GSS initiate failed" when trying to use the driver to connect to a Kerberos-enabled cluster. Typically, Java vendors only allow encryption strength up to 128 bits by default. If you are using greater encryption strength in your environment (for example, 256-bit encryption), then you might encounter this error.

Diagnosing the Issue

If you encounter the error message "Unable to connect to server: GSS initiate failed", confirm that it is occurring due to encryption strength by enabling Kerberos layer logging in the JVM and then checking if the log output contains the error message "KrbException: Illegal key size".

To enable Kerberos layer logging in a Sun JVM:

- Choose one:
  - In the Java command you use to start the application, pass in the following argument:

    ```
    -Dsun.security.krb5.debug=true
    ```

  - Or, add the following code to the source code of your application:

    ```
    System.setProperty("sun.security.krb5.debug","true")
    ```

To enable Kerberos layer logging in an IBM JVM:

- Choose one:
  - In the Java command you use to start the application, pass in the following arguments:

    ```
    -Dcom.ibm.security.krb5.Krb5Debug=all
    -Dcom.ibm.security.jgss.debug=all
    ```

  - Or, add the following code to the source code of your application:

    ```
    System.setProperty
    ("com.ibm.security.krb5.Krb5Debug","all");
    System.setProperty
    ("com.ibm.security.jgss.debug","all");
    ```

Resolving the Issue

After you confirm that the error is occurring due to encryption strength, you can resolve the issue by downloading and installing the *Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files* extension from your Java vendor. Refer to the instructions from the vendor to install the files to the correct location.

> **! Important:**
>
> Consult your company's policy to make sure that you are allowed to enable encryption strengths in your environment that are greater than what the JVM allows by default.

If the issue is not resolved after you install the JCE policy files extension, then restart your machine and try your connection again. If the issue persists even after you restart your machine, then verify which directories the JVM is searching to find the JCE policy files extension. To print out the search paths that your JVM currently uses to find the

JCE policy files extension, modify your Java source code to print the return value of the following call:

```
System.getProperty("java.ext.dirs")
```

Configuring SSL

Note:

In this documentation, "SSL" indicates both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The driver supports industry-standard versions of TLS/SSL.

If you are connecting to a Hive server that has Secure Sockets Layer (SSL) enabled, you can configure the driver to connect to an SSL-enabled socket. When connecting to a server over SSL, the driver uses one-way authentication to verify the identity of the server.

One-way authentication requires a signed, trusted SSL certificate for verifying the identity of the server. You can configure the driver to access a specific TrustStore or KeyStore that contains the appropriate certificate. If you do not specify a TrustStore or KeyStore, then the driver uses the default Java TrustStore named `jssecacerts`. If `jssecacerts` is not available, then the driver uses `cacerts` instead.

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

To configure SSL:

1. Set the `SSL` property to `1`.
2. If you are not using one of the default Java TrustStores, then do one of the following:
   - Create a TrustStore and configure the driver to use it:
     a. Create a TrustStore containing your signed, trusted server certificate.
     b. Set the `SSLTrustStore` property to the full path of the TrustStore.
     c. Set the `SSLTrustStorePwd` property to the password for accessing the TrustStore.
   - Or, create a KeyStore and configure the driver to use it:
     a. Create a KeyStore containing your signed, trusted server certificate.
     b. Set the `SSLKeyStore` property to the full path of the KeyStore.
     c. Set the `SSLKeyStorePwd` property to the password for accessing the KeyStore.
3. Optionally, to allow the SSL certificate used by the server to be self-signed, set the `AllowSelfSignedCerts` property to `1`.

> **Important:**
>
> When the `AllowSelfSignedCerts` property is set to `1`, SSL verification is disabled. The driver does not verify the server certificate against the trust store, and does not verify if the server's host name matches the common name or subject alternative names in the server certificate.

4. Optionally, to allow the common name of a CA-issued certificate to not match the host name of the Hive server, set the `CAIssuedCertNamesMismatch` property to `1`.

For example, the following connection URL connects to a data source using username and password (LDAP) authentication, with SSL enabled:

```
jdbc:hive2://localhost:10000;AuthMech=3;SSL=1;
SSLKeyStore=C:\\Users\\bsmith\\Desktop\\keystore.jks;SSLKeyS
torePwd=simbaSSL123;UID=hs2;PWD=simba123
```

> **Note:**
>
> For more information about the connection properties used in SSL connections, see Driver Configuration Options on page 45.

Configuring Server-Side Properties

You can use the driver to apply configuration properties to the Hive server by setting the properties in the connection URL.

For example, to set the `mapreduce.job.queuename` property to `myQueue`, you would use a connection URL such as the following:

```
jdbc:hive://localhost:18000/default2;AuthMech=3;
UID=simba;PWD=simba;mapreduce.job.queuename=myQueue
```

Note:

For a list of all Hadoop and Hive server-side properties that your implementation supports, run the `set -v` command at the Hive CLI command line or Beeline. You can also execute the `set -v` query after connecting using the driver.

Configuring Logging

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

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

In the connection URL, set the `LogLevel` key to enable logging at the desired level of detail. The following table lists the logging levels provided by the Simba Hive JDBC Driver, in order from least verbose to most verbose.

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

To enable logging:

1. Set the `LogLevel` property to the desired level of information to include in log files.
2. Set the `LogPath` property to the full path to the folder where you want to save log files. To make sure that the connection URL is compatible with all JDBC applications, escape the backslashes (`\`) in your file path by typing another backslash.

For example, the following connection URL enables logging level 3 and saves the log files in the `C:\temp` folder:

```
jdbc:hive://localhost:11000;LogLevel=3;LogPath=C:\\temp
```

3. To make sure that the new settings take effect, restart your JDBC application and reconnect to the server.

The Simba Hive JDBC Driver produces the following log files in the location specified in the `LogPath` property:

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

If the `LogPath` value is invalid, then the driver sends the logged information to the standard output stream (`System.out`).

To disable logging:

1. Set the `LogLevel` property to `0`.
2. To make sure that the new setting takes effect, restart your JDBC application and reconnect to the server.

Features

More information is provided on the following features of the Simba Hive JDBC Driver:

- SQL Query versus HiveQL Query on page 34
- Data Types on page 34
- Catalog and Schema Support on page 35
- Write-back on page 36
- IHadoopStatement on page 36
- IHadoopConnection on page 40
- Security and Authentication on page 43

SQL Query versus HiveQL Query

The native query language supported by Hive is HiveQL. HiveQL is a subset of SQL-92. However, the syntax is different enough that most applications do not work with native HiveQL.

Data Types

The Simba Hive JDBC Driver supports many common data formats, converting between Hive, SQL, and Java data types.

The following table lists the supported data type mappings.

Hive Type	SQL Type	Java Type
BIGINT	BIGINT	java.math.BigInteger
BINARY	VARBINARY	byte[]
BOOLEAN	BOOLEAN	Boolean
CHAR (Available only in Hive 0.13.0 or later)	CHAR	String
DATE	DATE	java.sql.Date

Hive Type	SQL Type	Java Type
DECIMAL (In Hive 0.13 and later, you can specify scale and precision when creating tables using the DECIMAL data type.)	DECIMAL	java.math.BigDecimal
DOUBLE	DOUBLE	Double
FLOAT	REAL	Float
INT	INTEGER	Long
SMALLINT	SMALLINT	Integer
TIMESTAMP	TIMESTAMP	java.sql.Timestamp
TINYINT	TINYINT	Short
VARCHAR (Available only in Hive 0.12.0 or later)	VARCHAR	String

The aggregate types (ARRAY, MAP, STRUCT, and UNIONTYPE) are not yet supported. Columns of aggregate types are treated as VARCHAR columns in SQL and STRING columns in Java.

Catalog and Schema Support

The Simba Hive JDBC Driver supports both catalogs and schemas to make it easy for the driver to work with various JDBC applications. Since Hive only organizes tables into schemas/databases, the driver provides a synthetic catalog named HIVE under which all of the schemas/databases are organized. The driver also maps the JDBC schema to the Hive schema/database.

Note:

Setting the `CatalogSchemaSwitch` connection property to `1` will cause Hive catalogs to be treated as schemas in the driver as a restriction for filtering.

Write-back

The Simba Hive JDBC Driver supports translation for the following syntax when connecting to a Hive Server 2 instance that is running Hive 0.14 or later:

- INSERT
- UPDATE
- DELETE
- CREATE
- DROP

If the statement contains non-standard SQL-92 syntax, then the driver is unable to translate the statement to SQL and instead falls back to using HiveQL.

IHadoopStatement

IHadoopStatement is an interface implemented by the driver's statement class. It provides access to methods that allow for asynchronous execution of queries and the retrieval of the Yarn ATS GUID associated with the execution.

The IHadoopStatement interface is defined by the `IHadoopStatement.java` file. This file should look like the following example:

```
//
==================================================================
=====================================
/// @file IHadoopStatement.java /// /// Exposed interface
for asynchronous query execution. /// /// Copyright (C) 2017
Simba Technologies Incorporated.
//
==================================================================
=====================================
package com.simba.hiveserver2.hivecommon.core;
import java.sql.ResultSet; import java.sql.SQLException;
import java.sql.Statement;
/**
* An interface that extends the standard SQL Statement
Interface but allows for asynchronous
* query execution.
* The polling for query execution will occur when {@link
ResultSet#next()} or
* {@link ResultSet#getMetaData()} is called.
*/ public interface IHadoopStatement extends Statement
```

```
{
   /**
      * Executes the given SQL statement asynchronously.
      * <p> * Sends the query off to the server but does not
      wait for query execution to complete.
      * A ResultSet with empty columns is returned.
      * The polling for completion of query execution is
      done when {@link ResultSet#next()} or
      * {@link ResultSet#getMetaData()}is called.
      * </p>
      *
      * @param sql
      An SQL statement to be sent to the database, typically
      a
      * static SQL SELECT statement.
      *
      * @return A ResultSet object that DOES NOT contain the
      data produced by the given query; never null.
      *
      * @throws SQLException If a database access error
      occurs, or the given SQL
      * statement produces anything other than a single
      * <code>ResultSet</code> object.
      */
   public ResultSet executeAsync(String sql) throws
   SQLException;
   /**
      * Returns the Yarn ATS guid.
      *
      * @return String The yarn ATS guid from the operation
      if execution has started,
      * else null.
      */
   public String getYarnATSGuid(); }
```

The following methods are available for use in IHadoopStatement:

- `executeAsync(String sql)`

  The driver sends a request to the server for statement execution and returns immediately after receiving a response from the server for the execute request without waiting for the server to complete the execution.

The driver does not wait for the server to complete query execution unless `getMetaData()` or `next()` APIs are called.

Note that this feature does not work with prepared statements.

For example:

```
import
com.simba.hiveserver2.hivecommon.core.IHadoopStatement;

public class TestExecuteAsyncClass
{
   public static void main(String[] args) throws
   SQLException
   {
      // Create the connection object.
      Connection connection = DriverManager.getConnection
      ("jdbc:hive2://localhost:10000");

      // Create the statement object.
      Statement statement = connection.createStatement();

      // Unwrap the java.sql.Statement object to an
      implementation of IHadoopStatement so the
      // execution can be done asynchronously.
         //
         // The driver will return from this call as soon as
         it gets a response from the
         // server for the execute request without waiting
         for server to complete query execution.
         ResultSet resultSet =
            statement.unwrap(
               IHadoopStatement.class).executeAsync(
            "select * from example_table");

      // Calling getMetaData() on the ResultSet here will
      cause the driver to wait for the server
      // to complete query execution before proceeding with
      the rest of the operation.
      ResultSetMetaData rsMetadata = resultSet.getMetaData
      ();
```

```
      // Excluding code for work on the result set
      metadata...

      // Calling getMetaData() on the ResultSet here, and if
      getMetaData() was not call prior to
      // this, will cause the driver to wait for the server
      to complete query execution before
      // proceeding with the rest of the operation.
      resultSet.next();

      // Excluding code for work on the result set ...
   }
}
```

- `getYarnATSGuid()`

  Returns the Yarn ATS GUID associated with the current execution. Returns null if the Yarn ATS GUID is not available.

For example:

```
public class TestYarnGUIDClass
{
   public static void main(String[] args) throws
   SQLException
   {
      // Create the connection object.
      Connection connection = DriverManager.getConnection
      ("jdbc:hive2://localhost:10000");

      // Create the statement object.
      Statement statement = connection.createStatement();

      // Execute a query.
      ResultSet resultSet = statement.executeQuery("select *
      from example_table");

         // Unwrap the java.sql.Statement object to an
         implementation of IHadoopStatement to access the
         // getYarnATSGuid() API call.
      String guid = statement.unwrap(
           IHadoopStatement.class).getYarnATSGuid();
   }
```

```
}
```

IHadoopConnection

IHadoopConnection is an interface implemented by the driver's statement class. It provides access to methods that allow for the retrieval, deletion, and renewal of delegation tokens.

The IHadoopStatement interface is defined by the `IHadoopStatement.java` file. This file should look like the following example:

```
//
==============================================================
=====================================
/// @file IHadoopConnection.java
///
/// Exposed interface for the retrieval of delegation
tokens.
///
/// Copyright (C) 2017 Simba Technologies Incorporated.
//
==============================================================
=====================================
package com.simba.hiveserver2.hivecommon.core;
import java.sql.Connection;
import java.sql.SQLException;
/**
* An interface that extends the standard SQL Connection
Interface but allows for the
* retrieval/renewal/cancellation of delegation tokens.
*/
public interface IHadoopConnection extends Connection
{
   /**
      * Sends a cancel delegation token request to the
      server.
      *
      * @param tokenString The token to cancel.
      * @throws SQLException If an error occurs while
      sending the request.
      */
```

```
    public void cancelDelegationToken(String tokenString)
    throws SQLException;

    /**
    * Sends a get delegation token request to the server
    and returns the token as an
    * encoded string.
    *
    * @param owner The owner of the token.
    * @param renewer The renewer of the token.
    *
    * @return The token as an encoded string.
    * @throws SQLException If an error occurs while
    getting the token.
    */
    public String getDelegationToken(String owner, String
    renewer) throws SQLException;

    /**
    * Sends a renew delegation token request to the sever.
    *
    * @param tokenString The token to renew.
    * @throws SQLException If an error occurs while
    sending the request.
    */
    public void renewDelegationToken(String tokenString)
    throws SQLException;
  }
```

The following methods are available for use in IHadoopConnection:

- `getDelegationToken(String owner, String renewer)`

  The driver sends a request to the server to obtain a delegation token with the given owner and renewer.

  The method should be called on a Kerberos-authenticated connection.
- `cancelDelegationToken()`

  The driver sends a request to the server to cancel the provided delegation token.
- `renewDelegationToken()`

  The driver sends a request to the server to renew the provided delegation token.

The following is a basic code sample that demonstrates how to use the above functions:

```
public class TestDelegationTokenClass
{
   public static void main(String[] args) throws
   SQLException
   {
      // Create the connection object with Kerberos
      authentication.
      Connection kerbConnection =
      DriverManager.getConnection(
         "jdbc:hive2://localhost:10000;AuthMech=1;KrbRealm=Y
         ourRealm;KrbHostFQDN=sample.com;KrbServiceName=hiv
         e;");

      // Unwrap the java.sql.Connection object to an
      implementation
      // of IHadoopConnection so the methods for delegation
      token
      // can be called.
      String delegationToken = kerbConnection.unwrap
      (IHadoopConnection.class).getDelegationToken("owner_
      name", "renewer_name");

      // The token can then be used to connect with the
      driver.
      String tokenConnectionString =
      "jdbc:hive2://localhost:10000;AuthMech=6;DelegationTok
      en=" + delegationToken;
      Connection tokenConnection =
      DriverManager.getConnection(tokenConnectionString);

      // Excluding code for work with the tokenConnection
      ...

      // The original token (delegationToken) can be
      cancelled or renewed by unwrapping the
      java.sql.Connection object again to
      // an implementation of IHadoopConnection.

      // Renewing the token:
```

```
        kerbConnection.unwrap
        (IHadoopConnection.class).renewDelegationToken
        (delegationToken);

        // Cancelling the token:
        kerbConnection.unwrap
        (IHadoopConnection.class).cancelDelegationToken
        (delegationToken);
    }
}
```

Security and Authentication

To protect data from unauthorized access, some Hive data stores require connections to be authenticated with user credentials or the SSL protocol. The Simba Hive JDBC Driver provides full support for these authentication protocols.

> **Note:**
>
> In this documentation, "SSL" indicates both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The driver supports industry-standard versions of TLS/SSL.

The driver provides mechanisms that allow you to authenticate your connection using the Kerberos protocol, your Hive user name only, or your Hive user name and password. You must use the authentication mechanism that matches the security requirements of the Hive server. For information about determining the appropriate authentication mechanism to use based on the Hive server configuration, see Authentication Mechanisms on page 19. For detailed driver configuration instructions, see Configuring Authentication on page 13.

Additionally, the driver supports SSL connections with one-way authentication. If the server has an SSL-enabled socket, then you can configure the driver to connect to it.

It is recommended that you enable SSL whenever you connect to a server that is configured to support it. SSL encryption protects data and credentials when they are transferred over the network, and provides stronger security than authentication alone. For detailed configuration instructions, see Configuring SSL on page 29.

The SSL version that the driver supports depends on the JVM version that you are using. For information about the SSL versions that are supported by each version of Java, see "Diagnosing TLS, SSL, and HTTPS" on the Java Platform Group Product Management Blog: https://blogs.oracle.com/java-platform-group/entry/diagnosing_tls_ssl_and_https.

Note:

The SSL version used for the connection is the highest version that is supported by both the driver and the server, which is determined at connection time.

Driver Configuration Options

Driver Configuration Options lists and describes the properties that you can use to configure the behavior of the Simba Hive JDBC Driver.

You can set configuration properties using the connection URL. For more information, see Building the Connection URL on page 11.

> **Note:**
>
> Property names and values are case-sensitive.

AllowSelfSignedCerts

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies whether the driver allows the server to use self-signed SSL certificates.

- `1`: The driver allows self-signed certificates.

  > **Important:**
  >
  > When this property is set to `1`, SSL verification is disabled. The driver does not verify the server certificate against the trust store, and does not verify if the server's host name matches the common name in the server certificate.

- `0`: The driver does not allow self-signed certificates.

> **Note:**
>
> This property is applicable only when SSL connections are enabled.

AsyncExecPollInterval

Default Value	Data Type	Required
`10`	Integer	No

Description

The time in milliseconds between each poll for the asynchronous query execution status.

"Asynchronous" refers to the fact that the RPC call used to execute a query against Hive is asynchronous. It does not mean that JDBC asynchronous operations are supported.

> **Note:**
>
> This option is applicable only to HDInsight clusters.

AuthMech

Default Value	Data Type	Required
Depends on the `transportMode` setting. For more information, see transportMode on page 57.	Integer	No

Description

The authentication mechanism to use. Set the property to one of the following values:

- `0` for No Authentication.
- `1` for Kerberos.
- `2` for User Name.
- `3` for User Name And Password.
- `6` for Hadoop Delegation Token.

CAIssuedCertsMismatch

Default Value	Data Type	Required
0	Integer	No

Description

This property specifies whether the driver requires the name of the CA-issued SSL certificate to match the host name of the Hive server.

- 0: The driver requires the names to match.
- 1: The driver allows the names to mismatch.

Note:

This property is applicable only when SSL connections are enabled.

CatalogSchemaSwitch

Default Value	Data Type	Required
0	Integer	No

Description

This property specifies whether the driver treats Hive catalogs as schemas or as catalogs.

- 1: The driver treats Hive catalogs as schemas as a restriction for filtering.
- 0: Hive catalogs are treated as catalogs, and Hive schemas are treated as schemas.

DecimalColumnScale

Default Value	Data Type	Required
10	Integer	No

Description

The maximum number of digits to the right of the decimal point for numeric data types.

DefaultStringColumnLength

Default Value	Data Type	Required
`255`	Integer	No

Description

The maximum number of characters that can be contained in STRING columns. The range of `DefaultStringColumnLength` is `0` to `32767`.

By default, the columns metadata for Hive does not specify a maximum data length for STRING columns.

DelegationToken

Default Value	Data Type	Required
None	String	Yes, if `AuthMech` is set to `6` (Hadoop Delegation Token)

Description

A Hadoop delegation token for authentication.

This token must be provided to the driver in the form of a Base64 URL-safe encoded string. It can be obtained from the driver using the `getDelegationToken()` function, or by utilizing the Hadoop distribution `.jar` files.

DelegationUID

Default Value	Data Type	Required
None	String	No

Description

Use this option to delegate all operations against Hive to a user that is different than the authenticated user for the connection.

> **Note:**
>
> This option is applicable only when connecting to a Hive Server 2 instance that supports this feature.

httpPath

Default Value	Data Type	Required
`None`	String	Yes, if `transportMode=http`.

Description

The partial URL corresponding to the Hive server.

The driver forms the HTTP address to connect to by appending the `httpPath` value to the host and port specified in the connection URL. For example, to connect to the HTTP address `http://localhost:10002/cliservice`, you would use the following connection URL:

```
jdbc:hive2://localhost:10002;AuthMech=3;transportMode=http;h
ttpPath=cliservice;
UID=hs2;PWD=simba;
```

> **Note:**
>
> By default, Hive servers use `cliservice` as the partial URL.

KrbAuthType

Default Value	Data Type	Required
0	Integer	No

Description

This property specifies how the driver obtains the Subject for Kerberos authentication.

- `0`: The driver automatically detects which method to use for obtaining the Subject:
  1. First, the driver tries to obtain the Subject from the current thread's inherited AccessControlContext. If the AccessControlContext contains multiple Subjects, the driver uses the most recent Subject.
  2. If the first method does not work, then the driver checks the `java.security.auth.login.config` system property for a JAAS configuration. If a JAAS configuration is specified, the driver uses that information to create a LoginContext and then uses the Subject associated with it.
  3. If the second method does not work, then the driver checks the KRB5_CONFIG and KRB5CCNAME system environment variables for a Kerberos ticket cache. The driver uses the information from the cache to create a LoginContext and then uses the Subject associated with it.
- `1`: The driver checks the `java.security.auth.login.config` system property for a JAAS configuration. If a JAAS configuration is specified, the driver uses that information to create a LoginContext and then uses the Subject associated with it.
- `2`: The driver checks the KRB5_CONFIG and KRB5CCNAME system environment variables for a Kerberos ticket cache. The driver uses the information from the cache to create a LoginContext and then uses the Subject associated with it.

KrbHostFQDN

Default Value	Data Type	Required
None	String	Yes, if `AuthMech=1`.

Description

The fully qualified domain name of the Hive Server 2 host.

KrbRealm

Default Value	Data Type	Required
Depends on your Kerberos configuration	String	No

Description

The realm of the Hive Server 2 host.

If your Kerberos configuration already defines the realm of the Hive Server 2 host as the default realm, then you do not need to configure this property.

KrbServiceName

Default Value	Data Type	Required
None	String	Yes, if `AuthMech=1`.

Description

The Kerberos service principal name of the Hive server.

LogLevel

Default Value	Data Type	Required
0	Integer	No

Description

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

! Important:

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

Set the property to one of the following numbers:

- `0`: Disable all logging.
- `1`: Enable logging on the FATAL level, which logs very severe error events that will lead the driver to abort.
- `2`: Enable logging on the ERROR level, which logs error events that might still allow the driver to continue running.
- `3`: Enable logging on the WARNING level, which logs events that might result in an error if action is not taken.
- `4`: Enable logging on the INFO level, which logs general information that describes the progress of the driver.
- `5`: Enable logging on the DEBUG level, which logs detailed information that is useful for debugging the driver.
- `6`: Enable logging on the TRACE level, which logs all driver activity.

When logging is enabled, the driver produces the following log files in the location specified in the `LogPath` property:

- A `HiveJDBC_driver.log` file that logs driver activity that is not specific to a connection.
- A `HiveJDBC_connection_[Number].log` file for each connection made to the database, where *[Number]* is a number that distinguishes each log file from the others. This file logs driver activity that is specific to the connection.

If the `LogPath` value is invalid, then the driver sends the logged information to the standard output stream (`System.out`).

LogPath

Default Value	Data Type	Required
The current working directory.	String	No

Description

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

PreparedMetaLimitZero

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies whether the `PreparedStatement.getMetadata()` call will request metadata from the server with `LIMIT 0`.

- `1`: The `PreparedStatement.getMetadata()` call uses `LIMIT 0`.
- `0`: The `PreparedStatement.getMetadata()` call does not use `LIMIT 0`.

PWD

Default Value	Data Type	Required
`anonymous`	String	Yes, if `AuthMech=3`.

Description

The password corresponding to the user name that you provided using the property UID on page 57.

! Important:

If you set the `AuthMech` to `3`, the default `PWD` value is not used and you must specify a password.

RowsFetchedPerBlock

Default Value	Data Type	Required
`10000`	Integer	No

Description

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

Any positive 32-bit integer is a valid value, but testing has shown that performance gains are marginal beyond the default value of 10000 rows.

SocketTimeout

Default Value	Data Type	Required
`0`	Integer	No

Description

The number of seconds that the TCP socket waits for a response from the server before raising an error on the request.

When this property is set to `0`, the connection does not time out.

SSL

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies whether the driver communicates with the Hive server through an SSL-enabled socket.

- `1`: The driver connects to SSL-enabled sockets.
- `0`: The driver does not connect to SSL-enabled sockets.

Note:

SSL is configured independently of authentication. When authentication and SSL are both enabled, the driver performs the specified authentication method over an SSL connection.

SSLKeyStore

Default Value	Data Type	Required
None	String	No

Description

The full path of the Java KeyStore containing the server certificate for one-way SSL authentication.

See also the property SSLKeyStorePwd on page 55.

Note:

The Simba Hive JDBC Driver accepts TrustStores and KeyStores for one-way SSL authentication. See also the property SSLTrustStore on page 56.

SSLKeyStorePwd

Default Value	Data Type	Required
None	Integer	Yes, if you are using a KeyStore for connecting over SSL.

Description

The password for accessing the Java KeyStore that you specified using the property SSLKeyStore on page 55.

SSLTrustStore

Default Value	Data Type	Required
`jssecacerts`, if it exists. If `jssecacerts` does not exist, then `cacerts` is used. The default location of `cacerts` is `jre\lib\security\`.	String	No

Description

The full path of the Java TrustStore containing the server certificate for one-way SSL authentication.

See also the property SSLTrustStorePwd on page 56.

Note:

The Simba Hive JDBC Driver accepts TrustStores and KeyStores for one-way SSL authentication. See also the property SSLKeyStore on page 55.

SSLTrustStorePwd

Default Value	Data Type	Required
None	String	Yes, if using a TrustStore

Description

The password for accessing the Java TrustStore that you specified using the property SSLTrustStore on page 56.

transportMode

Default Value	Data Type	Required
`sasl`	String	No

Description

The transport protocol to use in the Thrift layer.

- `binary`: The driver uses the Binary transport protocol.

  When connecting to a Hive Server 1 instance, you must use this setting. If you use this setting but do not specify the `AuthMech` property, then the driver uses `AuthMech=0` by default. This setting is valid only when the `AuthMech` property is set to `0` or `3`.
- `sasl`: The driver uses the SASL transport protocol.

  If you use this setting but do not specify the `AuthMech` property, then the driver uses `AuthMech=2` by default. This setting is valid only when the `AuthMech` property is set to `1`, `2`, or `3`.
- `http`: The driver uses the HTTP transport protocol.

  When connecting to Hive through the Apache Knox Gateway, you must use this setting. If you use this setting but do not specify the `AuthMech` property, then the driver uses `AuthMech=3` by default. This setting is valid only when the `AuthMech` property is set to `3`.

  If you set this property to `http`, then the port number in the connection URL corresponds to the HTTP port rather than the TCP port, and you must specify the `httpPath` property. For more information, see httpPath on page 49.

UID

Default Value	Data Type	Required
`anonymous`	String	Yes, if `AuthMech=3`.

Description

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

Important:

If you set the `AuthMech` to `3`, the default `UID` value is not used and you must specify a user name.

UseNativeQuery

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies whether the driver transforms the queries emitted by applications.

- `1`: The driver does not transform the queries emitted by applications, so the native query is used.
- `0`: The driver transforms the queries emitted by applications and converts them into an equivalent form in HiveQL.

Note:

If the application is Hive-aware and already emits HiveQL, then enable this option to avoid the extra overhead of query transformation.

zk

Default Value	Data Type	Required
None	String	No

Description

The connection string to one or more ZooKeeper quorums, written in the following format where *[ZK_IP]* is the IP address, *[ZK_Port]* is the port number, and *[ZK_Namespace]* is the namespace:

```
[ZK_IP]:[ZK_Port]/[ZK_Namespace]
```

For example:

```
jdbc:hive2://zk=192.168.0.1:2181/hiveserver2
```

Use this option to enable the Dynamic Service Discovery feature, which allows you to connect to Hive servers that are registered against a ZooKeeper service by connecting to the ZooKeeper service.

You can specify multiple quorums in a comma-separated list. If connection to a quorum fails, the driver will attempt to connect to the next quorum in the list.

Contact Us

If you have difficulty using the driver, please contact our Technical Support staff. We welcome your questions, comments, and feature requests.

Note:

To help us assist you, prior to contacting Technical Support please prepare a detailed summary of the client and server environment including operating system version, patch level, and configuration.

You can contact Technical Support via the Magnitude Support Community at http://magnitudesoftware.com/online-support/.

You can also follow us on Twitter @SimbaTech and @Mag_SW

Third-Party Trademarks

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Apache Hive, Apache, and Hive are trademarks or registered trademarks of The Apache Software Foundation or its subsidiaries in Canada, United States and/or other countries.

All other trademarks are trademarks of their respective owners.