INTERNAL
SAP HANA Client 2.0
Document Version: 1.1 – 2024-09-23

SAP HANA Client Interface Programming Reference

© 2024 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 SAP HANA Client Interface Programming Reference. 6

2 Connecting to SAP HANA Databases and Servers. 8

2.1 How to Use the Client Interfaces with SAP HANA Platform. 8

2.2 Using the Client Interfaces with SAP HANA Cloud. 10

Securing Connections to SAP HANA Cloud Using SAP Common Crypto Library. 11

2.3 How to Use the Client Interfaces with SAP HANA Service. 12

2.4 Reconnecting to the Server. 12

Reconnection Behaviour Changes. 14

Errors Related To Reconnect. 15

2.5 Setting Session-Specific Client Information. 16

2.6 Environment Variables. 20

3 Configuring Clients for Secure Connections. 28

3.1 Server Certificate Authentication. 28

3.2 Configuring the Client for Client-Side Encryption and LDAP. 31

3.3 HTTP Proxy Client Connections. 32

Implement HTTP Proxy Client Connections. 33

3.4 Mutual Authentication (SAP HANA Platform). 34

Implement Mutual Authentication (SAP HANA Platform). 34

4 SAP HANA Application Programming Interfaces. 40

4.1 ADO.NET Application Programming. 40

The SAP HANA Provider for ADO.NET. 41

The .NET Core Provider. 74

The Microsoft .NET Framework Provider. 75

SAP HANA Data Provider for ADO.NET API Reference. 109

4.2 Go (golang) Application Programming. 398

Go (golang) Differences from Other Database Vendors. 400

Set Up Your Application to Use the Go Driver Package. 401

Build and Run the Go Examples. 403

Connect from Go to SAP HANA. 405

Driver-Specific Extensions. 427

Uninstall the Go Driver. 436

4.3 JDBC Application Programming. 436

Connect to SAP HANA via JDBC. 437

Cursor Holdability Support. 470

JDBC Data Source Example for Use with Apache Tomcat. 471

JDBC Driver Static Methods. . . . 476
Savepoint Support. . . . 490
JDBC Tracing and Trace Options. . . . 491
Hibernate Dialect for SAP HANA. . . . 497
4.4 Node.js Application Programming. . . . 498
Configure the Node.js Driver (Client Install). . . . 502
Run the Node.js Examples. . . . 503
Encrypting Node.js Connections. . . . 504
Node.js Connection Properties. . . . 506
Node.js Connection Pooling. . . . 530
Node.js Tracing. . . . 533
Node.js TypeScript Support. . . . 535
Connection Class. . . . 536
ConnectionPool Class. . . . 554
Hana Class. . . . 561
ResultSet Class. . . . 567
Statement Class. . . . 583
Stream Module. . . . 600
Promise Module. . . . 608
4.5 ODBC Application Programming. . . . 611
Connect to SAP HANA via ODBC. . . . 611
ODBC Statement Attributes. . . . 645
ODBC Tracing and Trace Options. . . . 656
4.6 Python Application Programming. . . . 665
Install the Python Driver. . . . 665
Connect to SAP HANA from Python. . . . 667
Uninstall the Python Driver. . . . 668
Connect Method and Python Connection Properties. . . . 668
Connection Class. . . . 691
Cursor Class. . . . 698
LOB Class. . . . 728
SQLAlchemy Dialect for SAP HANA. . . . 732
4.7 Ruby Application Programming. . . . 732
Configure Ruby for SAP HANA. . . . 733
Configure Ruby on Rails Support. . . . 734
Run Ruby Examples. . . . 736
Ruby Connection Properties. . . . 737
SAP HANA Ruby API Reference. . . . 754
4.8 SQLDBC Library. . . . 783
SQLDBC Tracing. . . . 785
SQLDBC Connection Properties. . . . 795

5 Client Support for Active/Active (Read Enabled) (SAP HANA Platform). 816
5.1 Connecting Using Active/Active (Read Enabled) (SAP HANA Platform). 816
5.2 Forced Statement Routing to a Site for Active/Active (Read Enabled) (SAP HANA Platform). 817
5.3 Hint-Based Statement Routing for Active/Active (Read Enabled) (SAP HANA Platform). 817

6 Application Tuning. 820
6.1 Tuning an Application. 820
6.2 Performance Tuning. 821
Packet Size. 821
Prepared Statement Caching. 823
Result Set Prefetch. 826
Control maxLazyDroppedStatements. 826
6.3 Data Transfer Tuning. 827
Network Compression. 827
Reducing the Amount of Data Transferred on the Network. 830
Reducing the Number of Requests. 831
6.4 Driver-Specific Tuning. 831
Best Practices for Node.js. 831
Best Practices for JDBC. 833
6.5 Memory Tuning. 834
Turning off Packet Caching. 834
Increasing Packet Size. 835

7 Statement Routing (SAP HANA Platform). 836
7.1 Client-side Statement Routing for Hash Partitioning. 839
7.2 Client-side Statement Routing for Range Partitioning (SAP HANA Platform). 839
7.3 Configuring Statement Routing via SQL Hints (SAP HANA Platform). 840
7.4 Determining Statement Routing Details (SAP HANA Platform). 840
7.5 Enabling SAP HANA Distribution Modes (SAP HANA Platform). 841
7.6 Backoff Timer for Regular Statement Routing (SAP HANA Platform). 844

8 SAP HANA Differences from Other Vendors. 845
8.1 Batch Inserts with Autocommit Enabled. 845
8.2 Character Encoding. 845
8.3 String to VARBINARY, ARRAY, and BLOB Conversions. 846
8.4 Timestamps in SAP HANA. 846

9 SAP HANA User Store (hdbuserstore). 848
9.1 hdbuserstore Commands. 851
9.2 Change the User Store Encryption Key. 855
9.3 Use the User Store (hdbuserstore). 856

10 SAP HANA HDBSQL (Command-Line Reference). 858

10.1 SAP HANA HDBSQL Options. . . . 858
10.2 Log On to a Database. . . . 870
10.3 Run Commands. . . . 871
10.4 Run Long Commands in Multiple-Line Mode. . . . 873
10.5 Edit Long Commands in an External File. . . . 874
10.6 Redirect Results to a File. . . . 875
10.7 Substitution Variables. . . . 876
10.8 Implement HTTP Proxy Client Connections for SAP HANA HDBSQL (SAP HANA Cloud). . . . 878

11 SAP HANA Calculation View API. . . . 879
11.1 Install the SAP HANA Calculation View API. . . . 879
11.2 Getting Started with Calculation Views. . . . 880
11.3 API Object Methods. . . . 882
11.4 Convert to Java Code. . . . 883
11.5 Troubleshooting. . . . 883
11.6 Limitations and Unsupported Use. . . . 884

12 SAP HANA Machine Learning APIs. . . . 885
12.1 R Machine Learning Client for SAP HANA. . . . 885
Configure the R Machine Learning Client for SAP HANA. . . . 885
12.2 Python Machine Learning Client for SAP HANA. . . . 886
Configure Python Machine Learning Client for SAP HANA. . . . 887

13 Important Disclaimer for Features in SAP HANA. . . . 889

1 SAP HANA Client Interface Programming Reference

This guide describes how to use the SAP HANA database client for connecting applications.

SAP HANA exposes data with client and web-based interfaces.

SAP HANA supports many common database application programming interfaces (APIs).

- SAP HANA supports forward-only, insensitive, non-updatable cursors across all interfaces.
- Client interfaces are available as long as the SAP HANA client is installed. The following APIs are supported:

API	Description	Has SQLDBC as a Basis
ADO.NET	Microsoft ADO.NET is the data access services component of the Microsoft .NET Framework and .NET Core application development. The Microsoft .NET Framework and .NET Core have separate SAP HANA providers in the `ado.net` and `dotnetcore` directories respectively. .NET Core applications require installation of the .NET Core Runtime that is available from Microsoft (https://dotnet.microsoft.com/download/dotnet-core/	Yes
Go	The Go (golang) driver.	Yes
JDBC	The JDBC driver.	No
Node.js	The Node.js driver for JavaScript.	Yes
ODBC	The ODBC driver.	Yes
Python DB API	The Python DB API (PyDBAPI).	Yes
Ruby	The Ruby language driver.	Yes
SQLDBC	The SQLDBC API (for internal use only).	Not applicable

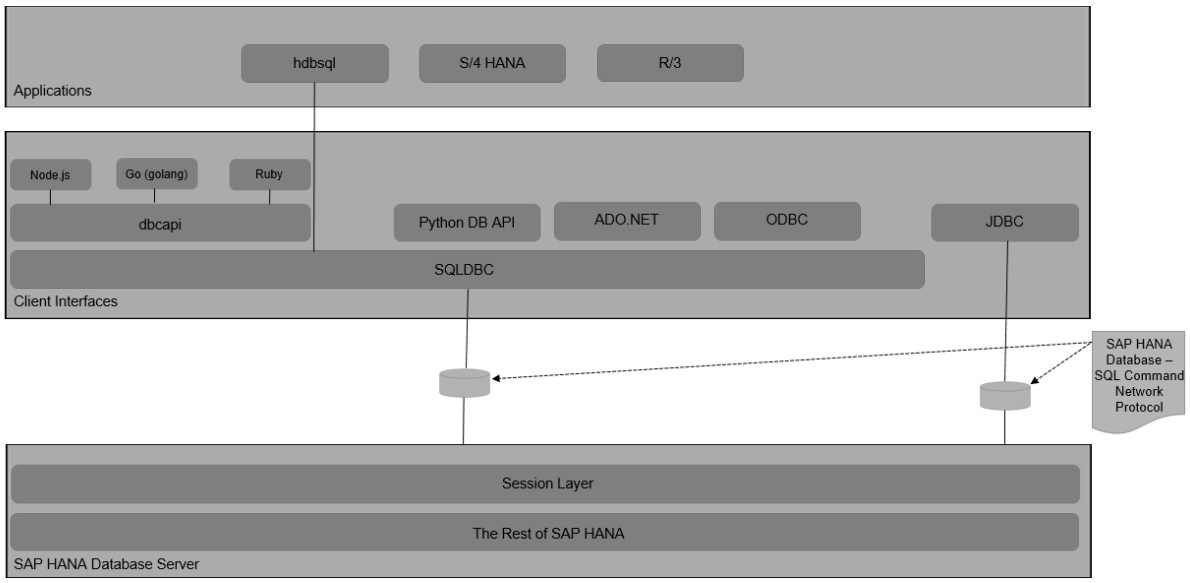
> **Note**
>
> For a list of supported API versions, see 3165810

- Web-based interfaces must be defined by the application developer, who determines what data to expose and to whom. The following web-based interfaces are supported:
  - OData
  - XMLA
  - Server-Side JavaScript

Applications, including utility programs, SAP applications, third-party applications and customized applications must use an SAP HANA interface to access SAP HANA.

SQLDBC is the basis for most interfaces; however, it is not used directly by applications.

The following diagram displays all supported SAP HANA interfaces and common SAP applications and utility programs:



2 Connecting to SAP HANA Databases and Servers

Learn how to connect to SAP HANA databases and servers.

To find out information regarding SAP HANA client version compatibilities, see 1906576.

User IDs and Passwords

Specify a specific user ID and password to the SAP HANA server by defining both the user ID (`<UID>`) and password (`<PWD>`) parameters in the client connection string.

Related Information

Connect to SAP HANA via ODBC [page 611]
Connect to SAP HANA via JDBC [page 437]

2.1 How to Use the Client Interfaces with SAP HANA Platform

There are some differences when using the client interfaces with SAP HANA Platform.

The examples below use ODBC connection strings, but the principle of specifying the server address and the database name applies to all SAP HANA client interfaces. For complete ODBC connection strings, you must add the `<driver>` or `<DSN>` connection parameters to the example connections on this page.

For the SAP HANA Platform, server ports are configured as follows:

- For the system database (SYSTEMDB), the port takes the following form: 3`<instance-number>`13. For example, 30013 (if the instance is 00). Additionally, specify the connection property `databaseName` to be automatically redirected to a specific database tenant running on the SAP HANA system.
- If you install a new system, you automatically get one tenant, unless you install an empty system. The port for the automatically created first tenant typically takes the following form: 3`<instance-number>`15. For example, 30115 (if the instance is 01).
- Additional tenants in the same system are on different ports. The second tenant is usually assigned to port 3`<instance-number>`41. Each additional tenant gets a new three-port range (three ports are used internally by each tenant, only one of these is available for client connections). For example, the port after 3`<instance-number>`41 is 3`<instance-number>`44.

- If you are connecting to an SAP HANA system that is version 2.0 SPS 00 or earlier that is a single-container system, then there is no SYSTEMDB and the port of the single database is 3<instance-number>15. For example, 32015 (if the instance is 20). In this case, the <databaseName> parameter should not be used.

If your SAP HANA Platform system is a multi-node system or contains a standby node, then you can specify multiple comma-separated addresses for the server location. For example:

```
serverNode=myServer:30013,myServerStandby:30013
```

Connecting to SAP HANA Servers

When connecting to SAP HANA, you must specify the server's address, including its port. You always connect to a database when connecting to a server.

The server's address is a TCP/IP address (a hostname or IP address) and a port and is often specified with the serverNode connection parameter, but this can vary between client interfaces. IPv4 and IPv6 addresses are supported. For example: "serverNode=192.0.2.104:30013" or "serverNode=[2001:db8::1a2b]:32013".

Connecting to the System Database (SYSTEMDB)

For system administration and monitoring tasks, connect to the SQL port of the system database (SYSTEMDB). For example, for instance 00, specify the following connection string:

```
serverNode=myserver:30013;UID=SYSTEM;PWD=systemPassword1;databaseName=SYSTEMDB
```

In this case, the databaseName parameter is optional.

Connecting to the Specified Database

To connect to a specified database, you can connect by specifying the port for SYSTEMDB and the databaseName. This is recommended over specifying the specific port for the database. For example:

```
serverNode=myserver:30013;UID=DB1USER;PWD=myPassword1;databaseName=DB1
```

Connecting to a Database at a Specific Port

To connect to a database at a specific port, the databaseName parameter is not required. For example:

```
serverNode=myserver:30013;UID=SYSTEM;PWD=systemPassword1
```

If the databaseName parameter is specified, then the client ensures that the database connected to matches databaseName.

2.2 Using the Client Interfaces with SAP HANA Cloud

Use the SAP HANA client interfaces to securely connect to the SAP HANA Cloud database.

Basic Connectivity

To connect to the SAP HANA Cloud, provide the user ID, password, and SAP HANA Cloud SQL Endpoint details.

The following example uses an ODBC-style connection string format to connect to the SAP HANA Cloud:

```
serverNode=12345678-
abcd-12ab-34cd-1234abcd.hana.hanacloud.ondemand.com:443;UID=myuser;PWD=mypassword
```

Supported Client Versions

It is recommended to use the latest version of the SAP HANA client interfaces available to connect to the SAP HANA Cloud. As newer versions of SAP HANA Cloud are released, a newer version of SAP HANA client is sometimes required to use the latest features. Minimum client versions for compatibility with SAP HANA Cloud features are described in SAP Note 3082128

Securing Client Connections to SAP HANA Cloud

All connections to SAP HANA Cloud are encrypted with Transport Layer Security (TLS).

SQLDBC By default, the SAP HANA Client uses either the OpenSSL library or Microsoft CryptoAPI to perform TLS encryption. The DigiCert Global Root G2 certificate is already packaged with the default included OpenSSL CA certificates or in the Microsoft Windows Certificate Manager included with the operating system.

If OpenSSL or Microsoft CryptoAPI is not available or if specific client features are required, the SAP Common Crypto Library (CommonCryptoLib) can also be used. For further details using CommonCryptoLib, see Securing Connections to SAP HANA Cloud Using SAP Common Crypto Library [page 11].

JDBC JDBC uses the TLS implementation in the Java Virtual Machine (JVM). Typically, the JVM includes a copy of the DigiCert Global Root G2 certificate in the default Java Keystore so no additional set up is required. If the JVM does not have the certificate available, it may need to be installed to the keystore using the Java keytool utility. For further details using the keytool utility, see Server Certificate Authentication [page 28].

Related Information

SAP HANA Cloud
SAP Note 3397584
SAP Note 3327214
SAP Note 3399572
SAP Note 2769719
SAP Note 3082128
SAP Note 3165810
DigiCert Trusted Root Authority Certificates
Server Certificate Authentication [page 28]

2.2.1 Securing Connections to SAP HANA Cloud Using SAP Common Crypto Library

Use the SAP Common Crypto Library (CommonCryptoLib) to secure the connection between the SAP HANA client and SAP HANA Cloud.

Procedure

1. Download the `DigiCert Global Root G2` certificate from https://dl.cacerts.digicert.com/DigiCertGlobalRootG2.crt.pem .

   > **Note**
   >
   > You can find details about root certificate updates on the DigiCert website at https://knowledge.digicert.com/general-information/digicert-root-and-intermediate-ca-certificate-updates-2023 . Also see SAP Note 3397584, SAP Note 3327214, and SAP Note 3399572.

2. Add the trusted root certificate file to a PSE file called `$SECUDIR/sapcli.pse` to be used by the client:
   a. Use the following command if `$SECUDIR/sapcli.pse` already exists:

      ```
      sapgenpse maintain_pk -p sapcli.pse -a DigiCertGlobalRootG2.crt.pem
      ```

   b. Use the following command if there is no existing `$SECUDIR/sapcli.pse`:

      ```
      sapgenpse gen_verify_pse -p sapcli.pse -a DigiCertGlobalRootG2.crt.pem
      ```

3. Specify the HANA Cloud host, UserID, and password. The following is an example for ODBC:

   ```
   serverNode=12345678-
   abcd-12ab-34cd-1234abcd.hana.hanacloud.ondemand.com:443;UID=myuser;PWD=mypassw
   ord
   ```

Related Information

SAP HANA Cloud
SAP Note 2769719
SAP Note 3397584
SAP Note 3327214
SAP Note 3399572
DigiCert Trusted Root Authority Certificates
Server Certificate Authentication [page 28]

2.3 How to Use the Client Interfaces with SAP HANA Service

There are some differences when using the client interfaces with SAP HANA Service. Refer to the client interfaces documentation for SAP HANA Service.

Related Information

SAP BTP, SAP HANA Service: SAP HANA Client Installation and Update Guide
SAP BTP, SAP HANA Service: SAP HANA Client Interface Programming Reference

2.4 Reconnecting to the Server

SQLDBC and JDBC have a built in `reconnect` functionality that attempts to restore a connection that was dropped by the network or server.

Disable this operation by specifying `reconnect=FALSE`.

> **Note**
>
> The `reconnect` behaviour is affected by the Transparent Session Recovery (TSR) feature. The TSR feature is also known as "Invisible Takeover."

Conditions to Reconnect Silently

The client reconnects after a send or receive error occurs in response to a dropped connection. Silent `reconnect`, which is a transparent `reconnect` with no errors that redoes the operation that got the send/receive error, can only occur if all of the following conditions are met:

- `reconnect` is enabled (default).
- The default isolation level is used.
- There is no transaction or, if there is a transaction, it is readonly.
- The server indicates that silent `reconnect` is allowed and that the transaction is recoverable. This is part of the TSR feature. In the following cases, the server does not allow a silent `reconnect` due to the session not being recoverable:
  - When there are global temporary table rows for the current connection.
  - If the SET TRANSACTION DDL OFF statement was executed on the connection.
- The operation that failed supports `reconnect`.

  > **Note**
  >
  > Many, but not all, common operations (including the Close Cursor, PREPARE, COMMIT, ROLLBACK, and most forms of the EXECUTE statements) do support `reconnect`. A reconnection also occurs when a SELECT query fails during a receive and the connection property `allowReconnectOnSelect` is enabled.
  >
  > One example of a common operation that does not support `reconnect` is the fetch operation. Once an application starts fetching through one or more result sets that were created before a reconnection, fetching after the reconnection results in an error because cursors cannot be restored by silent `reconnect`. However, the application may be able to successfully fetch a few rows after the reconnection if the client already fetched those rows from the server before it was reconnected.

- The error was detected on the send of the request, not on the receive of the response.
- The original authentication for the connection is currently valid. For example, the password has not changed since the connection was established.
- For JDBC only, the send or receive error is not due to a client-detected timeout configured by the `communicationTimeout` connection property or SAP HANA Cloud heartbeat feature.

> **Note**
>
> If there is a secondary session fallback and only an internal routed connection is dropped, the only behaviour change is an error if there were open result sets on the dropped internal routed connection that the application continues to fetch from.

Transparent Session Recovery (TSR)

TSR allows HANA System Replication (HSR) failovers and short server downtimes (crashes and restarts, for example) to reconnect without errors. If the client needs to reconnect to the server due a dropped connection and TSR is enabled, the client attempts to reconnect for a server-configurable period of time, with a default of 600 seconds.

Configure TSR by specifying `indexserver [session] enable_session_recovery` as TRUE or FALSE.

During a `reconnect`, SQLDBC tries to reestablish a connection within the time given by the `communicationTimeout` option set by the application or the server's TSR time, whichever is less.

Tracing

If it is unclear why the client is reconnecting or why the client cannot silently reconnect, tracing may provide more information.

For SQLDBC, enable DISTRIBUTION tracing to provide additional information. DISTRIBUTION tracing can be enabled for hours without a noticeable performance impact.

For JDBC, enable the CONNECTIONS and DISTRIBUTION tracing options to provide additional information.

Related Information

SQLDBC Connection Properties [page 795]
Connecting to SAP HANA Databases and Servers [page 8]
SQL Statements
SQLDBC Trace Options [page 786]
JDBC Tracing and Trace Options [page 491]

2.4.1 Reconnection Behaviour Changes

Certain behaviour changes may occur after a `reconnect` takes place.

If a `reconnect` occurs, session variables and most connection settings, such as isolation level, autocommit, and current schema settings, are restored and prepared statements are still valid.

The following states are not restored after a silent `reconnect` and/or a `reconnect` after a "Connection down" error occurs:

- Temporary objects (such as local temporary tables and rows in global temporary tables).
- Result sets and cursors open at the time of the `reconnect`.
- The current transactional DDL setting.
- The connection ID.
- Switching the database user using the CONNECT statement.
  - For example, `CONNECT <new_user> PASSWORD <Password1>`.
- transaction_access_mode and transaction_lock_wait_timeout are reset to their default mode even though the application did not end the transaction.

> **Note**
>
> The only behaviour change of a secondary session fallback is that result sets and cursors on the dropped secondary session can get errors. None of the other behaviour changes apply to secondary session fallback.

Related Information

Connecting to SAP HANA Databases and Servers [page 8]
Invisible Takeover and Restart
SQLDBC Connection Properties [page 795]

2.4.2 Errors Related To Reconnect

The `reconnect` function is an attempt by the client to reduce the effects of network or server errors, when possible. However, errors may still occur.

If errors persist, the root problem is with the network or server dropping the connection. If a connection down error occurs and the application requires resiliency from dropped connections, the application handles the error by redoing all operations since the last commit or rollback and closing and re-executing any required cursors/result sets.

SQLDBC Errors

If the connection is dropped and the `reconnect` conditions have not been met, SQLDBC normally sets the error -10807 "Connection down ...". The exact text of this error can vary. For example:

```
Connection down: [89006] System call 'recv' failed, rc=104:Connection reset by
peer {<address>}
```

Due to the long TSR wait timeout, this error is returned immediately to the application without performing the TSR connect retry logic. If, after receiving this error, the application attempts to perform another operation on the connection (PREPARE, EXECUTE, etc) then the client attempts to reconnect at that point, including retrying due to the TSR wait timeout.

If the `connDownRollbackError` connection property is set to TRUE, and there is a write transaction in progress, SQLDBC sets the error -10806 "Connection down due to '`<details>`' and transaction rolled back" instead of the -10807 "Connection down ..." error.

If a reconnect occurs and an application later attempts to fetch from a cursor created before the `reconnect`, SQLDBC normally sets the error -10516 "Invalid result set because session was dropped and reconnected. The connection was dropped by the server, network or due to an internal error".

JDBC Errors

If the connection is dropped and the `reconnect` conditions are not met, JDBC returns the exception -708/08006 “Connection to database server lost; check server and network status.” In this case, JDBC does not reattempt to reconnect on future connections. Like SQLDBC, this error is returned immediately to the application without doing the TSR connect retry logic.

If the connection is dropped and the `reconnect` conditions have been met, but the `reconnect` attempt fails, JDBC returns the same above-mentioned exception. In this case, JDBC reattempts to reconnect on future connections.

If the connection is dropped and silent `reconnect` is successful, no exception is returned. However, if an application later attempts to fetch from a cursor that was created before the reconnect, JDBC returns the same exception as above.

2.5 Setting Session-Specific Client Information

The client information is a list of session variables (defined in property-value pairs that are case sensitive) that an application can set in the SAP HANA client interface.

The SAP HANA client interface stores the values specified in the M_SESSION_CONTEXT system table. Apart from storing the client information in the appropriate place in the database, the methods described here do not alter the behavior of the connection in any way. The values supplied to these methods are used for internal checks, diagnostics, and debugging purposes only.

When connecting to your database, you can set session variables by using the `sessionVariable:` connection option prefix. Each session variable must be specified as an independent key-value pair.

For example, in JDBC:

```
jdbc:sap://ykfl00540545a:30115/?
autocommit=false&sessionVariable:APPLICATION=myapp&sessionVariable:APPLICATIONUSE
R=user1&sessionVariable:myvar=myval&distribution=connection
```

For example, in ODBC:

```
DSN=HANADataSource;UID=myUser;PWD=myPassword;sessionVariable:APPLICATION=myapp;se
ssionVariable:APPLICATIONUSER=user1;sessionVariable:myvar=myval
```

The following session variables are always set by the client:

Variable Name	Definition	Default (JDBC)	Default (other APIs)
APPLICATION	Displays the application name.	System.getProperty(<sun_java_command>)	The executable name (with `.exe` suffix removed on Microsoft Windows).

Variable Name	Definition	Default (JDBC)	Default (other APIs)
APPLICATIONUSER	Displays the application user name.	System.getProperty(<user_name>)	The operating system user name associated with the application process.
DRIVERVERSION	Displays the driver version.	System.getProperty(<driver_version>)	The operating system driver version associated with the application process.

The following example displays how to view a variable session with SAP HANA HDBSQL:

```
./hdbsql -n localhost:30015 -u system -p manager
"SELECT * FROM M_SESSION_CONTEXT WHERE KEY='DRIVERVERSION'"
HOST,PORT,CONNECTION_ID,KEY,VALUE,SECTION
"localhost",30003,301640,"DRIVERVERSION","2.4.9","USER"
```

ODBC

Use connection attributes to access client information from ODBC:

```
SQL_SQLDBC_SET_CLIENTINFO_KEY = 11003
SQL_SQLDBC_SET_CLIENTINFO_VALUE = 11004
SQL_SQLDBC_GET_CLIENTINFO_VALUE = 11005
```

The attributes are defined in `<client install directory>/sdk/odbc/incl/sqlsdbodbc.h`, and are illustrated in the following example:

```
// setting key HOMER and value mafimi
char key[] = "HOMER";
rc = SQLSetConnectAttr(hdbc, SQL_SQLDBC_SET_CLIENTINFO_KEY, key, SQL_NTS);
char value[] = "mafimi";
rc = SQLSetConnectAttr(hdbc, SQL_SQLDBC_SET_CLIENTINFO_VALUE, value, SQL_NTS);
// retrieving the value for the currently set SQL_SQLDBC_SET_CLIENTINFO_KEY key
// if the buffer is too short: rc is SQL_SUCCESS_WITH_INFO and sqlstate 01004:
// String data, right truncated
char buf[64];
SQLINTEGER bufLen = 0;
rc = SQLGetConnectAttr(hdbc, SQL_SQLDBC_GET_CLIENTINFO_VALUE, buf, sizeof(buf),
&bufLen);
```

JDBC

The JDBC 4.2 API provides the following methods on `java.sql.Connection` to `set` or `get` the client information for a session:

- `setClientInfo`

  ```
  void setClientInfo(String name,
                     String value)
  ```

```
                    throws SQLClientInfoException
void setClientInfo(Properties properties)
                    throws SQLClientInfoException
```

- `getClientInfo`

```
String getClientInfo(String name)
                      throws SQLException
Properties getClientInfo()
                      throws SQLException
```

> **Note**
>
> When you retrieve a client information property name with the command `getClientInfo`, the names are case sensitive.

For details on the API, refer to the JDBC specification:

```
//set client info property
connection.setClientInfo("APPLICATION", "Simpson");
connection.setClientInfo("APPLICATIONVERSION", "0.100");
//unset client info property
connection.setClientInfo("APPLICATION", null);
//get client info property
Properties ci = connection.getClientInfo();
Enumeration e = ci.keys();
while (e.hasMoreElements()) {
    String key = (String) e.nextElement();
    String val = (String) ci.get(key);
    System.out.println("Key="+key+" Value="+val);
}
```

Python DB

The Python DB API (`hdbcli.dbapi`) allows you to set client information during the initial connection.

- Example

```
conn = dbapi.connect(address=<host>, port=<port>, user=<user>,
password=<password>,
    sessionvariables={'APPLICATION':'MyPythonProg'})
```

The Python DB API also provides the following methods on the `Connection` class to `get` or `set` the client information:

- `getclientinfo`
  Gets client information with a key.

```
conn = dbapi.connect(address=<host>, port=<port>, user=<user>,
password=<password>)
value = conn.getclientinfo("MYKEY")
```

  - If the key for `getclientinfo` is "`key=None`", then **all** defined key-value pairs are returned.
  - The `dbapi.Connection.getclientinfo` command is similar to the command `java.sql.Connection.getClientInfo()` that is included in the JDBC API.
- `setclientinfo`

Sets client information with a key and a corresponding value.

```
conn = dbapi.connect(address=<host>, port=<port>, user=<user>,
password=<password>)
conn.setclientinfo("MYKEY", 1000)
```

- If the key for `setclientinfo` is "`value=None`", then the key is removed from the client-information dictionary.
- The `dbapi.Connection.setClientInfo` command is similar to the command `java.sql.Connection.setClientInfo()` that is included in the JDBC API.

- Example

```
conn = dbapi.connect(address=<host>, port=<port>, user=<user>,
password=<password>)
cur = conn.cursor()
conn.setclientinfo("SCRIPT", _file_)
cur.execute("create column table table_to_check_clientinfo (z int primary
key)")
ci = conn.getclientinfo("SCRIPT")
```

Node.js

The Node.js driver's connection class provides the setClientInfo and getClientInfo methods for setting and accessing the client information.

- For example:

```
var hana = require('hana.node');
var conn = hana.createConnection();
conn.connect('serverNode=myserver:30015;uid=system;pwd=manager');
conn.setClientInfo('locale', 'en-CA');
var locale = conn.getClientInfo('locale');
conn.close();
```

Go

Set session variables for the Go driver by using the `sessionVariable:` connection option prefix. Each session variable must be specified as an independent key-value pair. For example:

```
hdb://<user>:<password>@<host>:<port>?
sessionVariable:<var1>=<val1>&sessionVariable:<var2>=<val2>
```

In the above example, replace `<var1>` and `<var2>` with the name of session variable.

Related Information

JDBC Connection Properties [page 441]

ODBC Connection Properties [page 621]

2.6 Environment Variables

Use environment variables to store different kinds of information.

Client Installation Environment Variables

Use connection environment variables to store installation-related information.

Environment Variable Name	Value	Description
HDB_INSTALL_NOPATH	<value>	Skips the PATH modification on Microsoft Windows.

Connection Environment Variables

Use connection environment variables to store connection-related information.

Environment Variable Name	Description
SAPLOCALHOST	Acts as the override for the virtual host name when connecting to a database.
HDB_NETWORK_GROUP	Acts as an override to the SQLDBC `networkGroup` connection property.
HDB_USE_IDENT	Acts as an override to the environment variable override for the <host-identity> portion of the secure user store location.
HDB_USE_STORE_PATH	Force the absolute path or the relative path, if the value is not prefixed by a file path separator, of the user store. Other controls that have influence on the store location are silently ignored, except if HDB_USE_IDENT is also specified then accessing the store would result in an error.

JDBC Tracing Environment Variables

Use tracing environment variables to store tracing-related information for JDBC. The traces are collected until the program finished. It is not possible to dynamically switch the traces off.

Environment Variable Name	Value	Description
HDB_JDBC_TRACEFILE	<JDBC-trace-file-name>	Implicitly enables tracing for all JDBC connections established in the latest Java VM when combined with the HDB_JDBC_TRACEOPTIONS environment variable. Changes to the trace settings file do not affect traces that are established using either both the traceFile and traceOptions connection properties or environment variables.
HDB_JDBC_TRACEOPTIONS	A comma-separated list of any of the following trace options: CONNECTIONS, API, PACKET, DISTRIBUTION, STATISTICS, CLEANERS, DEBUG, TIMESTAMPS, and ELAPSEDTIMES	Implicitly enables tracing for all JDBC connections established in the current JVM when combined with the HDB_JDBC_TRACEFILE environment variable. Changes to the trace settings file do not affect traces that are established using either both the traceFile and traceOptions connection properties or environment variables.

SQLDBC Tracing Environment Variables

Use tracing environment variables to store tracing-related information for SQLDBC-based client interfaces.

Environment Variable Name	Value	Description
HDB_SQLDBC_TRACEFILE	<SQLDBC-trace-file-name>	Implicitly enables tracing for all SQLDBC connections in the latest process when combined with the HDB_SQLDBC_TRACEOPTS environment variable. The special file names "stdout" and "stderr" are supported to send trace information to stdout and stderr, respectively.
HDB_SQLDBC_TRACEOPTS	<trace-options>	Implicitly enables tracing for all SQLDBC-based client connections when combined with the HDB_SQLDBC_TRACEFILE environment variable. See the SQLDBC Tracing [page 785] topic in this guide.

Secure User Store (hdbuserstore) Environment Variables

Use the secure user store (`hdbuserstore`) environment variables to store your hdbuserstore in one environment so that users on all hosts can connect to the SAP HANA database using the same secure store information.

Environment Variable Name	Value	Default	Description
HDB_USE_STORE_PATH	<secure-store-path-location>		Specifies the environment variable override for the location of the secure user store file. If the HDB_USE_IDENT environment variable is also specified, then the hdbuserstore operation would result in an error instead of it being silently ignored.

Environment Variable Name	Value	Default	Description
HDB_USE_IDENT		<host-identity>	Specifies the environment variable override for the <host-identity> portion of the secure user store location.

Node.js Environment Variables

Environment Variable Name	Default	Description
HDB_NODEJS_INSTALL_PLATFORMS		Specifies which platforms keep their native platform libraries at npm install time. This is done using a comma-separated list with no spaces. The following platforms are currently installed: • darwinarm64-xcode12 • darwinintel64-xcode7 • linuxaarch64-gcc9 • linuxaarch64-gcc9 • linuxppc64le-gcc6 • linuxx86_64_gcc6 • linuxx86_64-gcc6 • ntamd64-msvc2015
HDB_NODE_PLATFORM_CLEAN		Removes platform-specific libraries at npm install time for platforms other than the current platform. To invoke this cleaning behavior, specify HDB_NODE_PLATFORM_CLEAN=1. This takes precedence over HDB_NODEJS_INSTALL_PLATFORMS. This is not recommended if you are vendoring dependencies.
HDB_NODEJS_THREADPOOL_SIZE	4	Specifies the number of worker threads in the thread pool.

ODBC Environment Variables

Use ODBC environment variables to store information gathered when using the ODBC API.

Environment Variable Name	Default	Description
`ODBCINI`	`$HOME/.odbc.ini`	Specifies the environment variable override for the full path and filename location of the `odbc.ini` file (ODBC custom configuration file) on Linux and UNIX. This file defines the ODBC data sources on Linux and UNIX.
`HOME`	This depends on the operating system.	Specifies the environment variable override for the path location of the `odbc.ini` file (ODBC Custom Configuration File) on Linux and UNIX. This file defines ODBC data sources on Linux and UNIX.

Crypto Environment Variables

Use crypto-related environment variables to store secure information.

Environment Variable Name	Value	Description
`SECUDIR`	The location of the CommonCryptoLib identity file (*.pse).	Specifies the environment variable override for the path location of the keystore file that contains the client's identity.
`HOME`	The location of the OpenSSL directory (*.ssl).	Specifies the environment variable override for the path location of the .ssl directory, used by OpenSSL to find OpenSSL identity files (key.pem, trust.pem).
`SAP_RETRIEVAL_PATH`	The location of OpenSSL ssl directory (*.ssl).	Specifies the environment variable override for the path location of an SSL directory, used to find OpenSSL identity files (key.pem, trust.pem).

Kerberos Environment Variables

Use Kerberos environment variables to store authentication information.

Environment Variable Name	Default	Description
KRB5_CONFIG	/etc/krb5.conf	Specifies the full path and file name location of the Kerberos configuration file.
KRB5_KTNAME	/etc/krb5.keytab	Specifies the full path and filename location of the Kerberos keytab file.

.NET Environment Variables

Use environment variables to store information regarding your .NET application.

Environment Variable Name	Default	Description
HDBADONET	C:\Program Files\SAP\hdbclient\ado.net\	Specifies the full path to the libadonetHDB.dll.
HDBDOTNETCORE	C:\Program Files\SAP\hdbclient\dotnetcore\ (Microsoft Windows) /usr/sap/hdbclient/dotnetcore (Linux and macOS)	Specifies the full path to the libadonetHDB.dll, libadonetHDB.so, or libadonetHDB.dylib file.

Library Environment Variables

Use environment variables to store information regarding loading libraries.

Environment Variable Name	Description
PATH	Specifies the library path on the Microsoft Windows operating system.
LD_LIBRARY_PATH	Specifies the library path on the Solaris operating system.

Environment Variable Name	Description
DYLD_LIBRARY_PATH	Specifies the library path on the macOS operating system.
LIBPATH	Specifies the library path on the AIX operating system.

SAP HANA HDBSQL Environment Variables

Use environment variables to store information related to the SAP HANA HDBSQL command-line application.

Environment Variable	Default	Description
HDBSQL_EDITOR	Notepad (Microsoft Windows) and vi on other operating systems.	The first environment variable to be checked to set the editor program for HDBSQL. On Windows, the variable must be set to the full path of an .exe file. Must be checked in order: HDBSQL_EDITOR, EDITOR, and VISUAL.
EDITOR	Notepad (Microsoft Windows) and vi on other operating systems.	The second environment variable to be checked to set the editor program for HDBSQL. On Windows, the variable must be set to the full path of an .exe file. Must be checked in order: HDBSQL_EDITOR, EDITOR, and VISUAL.
VISUAL	Notepad (Microsoft Windows) and vi on other operating systems.	The third environment variable to be checked to set the editor program for HDBSQL. On Windows, the variable must be set to the full path of an .exe file. Must be checked in order: HDBSQL_EDITOR, EDITOR, and VISUAL.
PAGER	Notepad (Microsoft Windows) and vi on other operating systems.	If set, this is used as the pager program for SAP HANA HDBSQL.

Related Information

SQLDBC Tracing [page 785]
Node.js Connection Properties [page 506]
ODBC Connection Properties [page 621]
Install the SAP HANA Client on UNIX, macOS, or Linux
Connect to SAP HANA via ODBC [page 611]
Use the User Store (hdbuserstore) [page 856]
SQLDBC Connection Properties [page 795]
SAP HANA User Store (hdbuserstore) [page 848]

3 Configuring Clients for Secure Connections

You can configure SAP HANA clients for secure client-server communication through TLS/SSL with certificate-based server authentication or mutual authentication.

Additionally, security can be enhanced through the use of client-side encryption or LDAP authentication.

Related Information

Secure Communication Between SAP HANA and JDBC/ODBC Clients

3.1 Server Certificate Authentication

When the SAP HANA client secures TLS connections with an SAP HANA server, the client verifies the server's certificate to ensure that it has connected to the correct server, thus preventing man-in-the-middle attacks.

To configure the SAP HANA client to authenticate the server certificate, perform the following steps:

1. Acquire the public certificate of the trusted root certificate authority.
   The client needs access to the public certificate of the certificate authority (CA) that signed the server's certificate. How this is done depends on how the server certificate was signed:

   Server certificate signed by a certificate authority

   If the server's certificate is signed by a certificate authority, the client needs to have the certificate authority's public root certificate. This can be extracted from the server's PSE file as follows:

   ```
   sapgenpse export_own_cert -r -p sapsrv.pse -o server_root.crt
   ```

   Alternately, if the certificate authority is a publicly recognized certificate authority, the public root certificate can be downloaded from their website.

   If connecting to SAP HANA Cloud, you need to trust the DigiCert Global Root CA and G2 certificate authorities, which can be downloaded from: https://www.digicert.com/digicert-root-certificates.htm

   The DigiCert Global Root CA certificate authority can be found in both DER and PEM format: https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt (DER) and https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem (PEM).

The DigiCert Global Root G2 certificate authority can be found in both DER and PEM format: https://dl.cacerts.digicert.com/DigiCertGlobalRootG2.crt (DER) and https://dl.cacerts.digicert.com/DigiCertGlobalRootG2.crt.pem (PEM).

> **Note**
>
> You can find details about root certificate updates on the DigiCert website at https://knowledge.digicert.com/general-information/digicert-root-and-intermediate-ca-certificate-updates-2023 . Also see SAP Note 3397584, SAP Note 3327214 and SAP Note 3399572.

Self-signed server certificate

If the server's certificate is self-signed, then the client needs to trust the server's public certificate itself. This can be extracted from the server's PSE file as follows:

```
sapgenpse export_own_cert -p sapsrv.pse -o server_root.crt
```

2. Provide the trusted root certificate to the client.
   Once the trusted root certificate for the server has been obtained, it must be provided to the client driver so it can connect to SAP HANA. How it is provided depends on which Cryptographic API provider you are using:

CommonCryptoLib

The trusted root certificate file needs to be converted to a PSE file to be used by the client when using the CommonCryptoLib provider:

```
sapgenpse gen_verify_pse -p sapcli.pse -a server_root.crt
```

> **Note**
>
> If `$SECUDIR/sapcli.pse` already exists use the following configuration:
>
> ```
> sapgenpse maintain_pk -p sapcli.pse -a server_root.crt
> ```

Optionally, you can specify a PIN for the PSE file. To avoid using a PIN, click *Enter* twice. To set a PIN, enter the following command, which also creates a credential file to access the PSE file:

```
sapgenpse seclogin -p ${SECUDIR} -x <PIN>
```

This creates a `${SECUDIR}/cred_v2` file that stores the PIN to the PSE file, which the HANA client reads when connecting.

By default, the driver expects the file name to be `sapcli.pse` and to be in the `$SECUDIR` directory. You can specify a PSE file with a different name or path with the `sslKeyStore` and `sslTrustStore` client configuration properties.

Set the connection property encrypt to TRUE to turn on TLS encryption or add the -e option if using SAP HANA HDBSQL.

OpenSSL (Linux, macOS, AIX)

The OpenSSL provider cannot use PSE files directly; it expects the public root certificate to be in PEM format.

To use the OpenSSL provider instead of the CommonCrypto provider, set the following client connection properties:

- Set the `sslCryptoProvider` client connection parameter to `openssl` so it uses the OpenSSL provider instead of the CommonCryptoLib provider.
- Set the `sslTrustStore` parameter to the file containing the server's root certificate authority that was created in Step 1 (server_root.crt in the example above), in PEM format.
- Set encrypt to TRUE to turn on encryption. For SAP HANA HDBSQL, adding the -e option is equivalent to setting encrypt to TRUE.

The following example uses SAP HANA HDBSQL to connect to the SAP HANA Cloud with server certificate authentication:

```
hdbsql -n 12345678-
abcd-12ab-34cd-1234abcd.hana.hanacloud.ondemand.com:443 -u
MyUser -p MyPassword123 -e -Z sslCryptoProvider=openssl -Z
sslTrustStore=DigiCertGlobalRootG2.crt.pem
```

Microsoft CryptoAPI (Microsoft Windows only)

The `sslTrustStore` connection property selects the System Store Location to use (see: https://docs.microsoft.com/en-us/windows/win32/seccrypto/system-store-locations). There are four options:

- MY
- Root
- Trust
- CA

By default, the client uses the MY system store. To trust a new certificate authority, add the trusted certificate authority certificate via the Microsoft Management Console Certificates Snap-in (`certmgr.msc`) to the desired System Store Location.

Java

The JDBC driver uses the TLS support that is included with the Java Development Kit (JDK). Use the Java keytool command to convert the server's root certificate to the Java keystore file format:

```
keytool -importcert -keystore server_root.jks -file
server_root.crt -storepass mypassword
```

Set the following connection properties to connect to an SAP HANA server with server certificate authentication:

- Set `trustStoreType` to `jks`
- Set `trustStore` to the file containing the server's root certificate authority (`server_root.jks` in the example above)
- Set `trustStorePassword` to the password used when creating the trust store (`mypassword` in the example above)
- Set encrypt to TRUE to turn on encryption

Considerations

To connect with TLS without any server authentication, set the `sslValidateCertificate` client connection property (`validateCertificate` for JDBC) to FALSE. This is not recommended as it exposes the connection to attacks by third parties when connecting over public or unsecured networks.

Use the `sslHostNameInCertificate` client connection property (`hostNameInCertificate` for JDBC) if the name in the server's certificate is not expected to match the hostname used by the client to connect to the server.

To ensure that SAP HANA uses the FQDN while connecting with TLS and that certificate validation for secured JDBC/ODBC connections is allowed, execute the following statement when connected to SYSTEMDB:

```
ALTER SYSTEM ALTER CONFIGURATION ('global.ini', 'SYSTEM') SET
('public_hostname_resolution', 'use_default_route') = 'fqdn' WITH RECONFIGURE;
```

Related Information

TLS/SSL Configuration on the SAP HANA Server
SAP HANA Platform: Secure Communication Between SAP HANA and JDBC/ODBC Clients
SAP HANA Cloud: Secure Communication Between SAP HANA Cloud and JDBC/ODBC Clients
SAP HANA Platform: Client-Side TLS/SSL Connection Properties (ODBC)
SAP HANA Cloud: Client-Side TLS Connection Properties (ODBC)
SAP HANA Platform: Client-Side TLS/SSL Connection Properties (JDBC)
SAP HANA Cloud: Client-Side TLS Connection Properties (JDBC)
Connect to SAP HANA Cloud via ODBC
Connect to SAP HANA Cloud via JDBC

3.2 Configuring the Client for Client-Side Encryption and LDAP

To use client-side encryption or LDAP authentication, ensure that your interface meets the following requirements.

SQLDBC For SQLDBC-based clients, you must use CommonCryptoLib to use client-side encryption.

Check the `/client` folder of the installation package to see if `COMMONCRYPTOLIB.TGZ` is present. If not, you can download CommonCryptoLib separately. For instructions on how to download CommonCryptoLib, see *Download and Install SAP Common Crypto Library* in the *SAP HANA Client Installation and Update Guide*.

JDBC	deferredPrepare Connection Property	The deferredPrepare connection property must be set to DISABLED (the default) to use client-side encryption.
	JDK Version	If you are using JDK 7, then use update 134 or later. Using JDK 7 versions with an earlier update number results in a key generation error.
	JDK Policy Files	Update the JDK policy files to support unlimited strength encryption. Failure to update the policy files results in a decryption error. Update the policy files as follows: • For JDK 6, download the files here:http://www.oracle.com/technetwork/java/javase/downloads/jce-6-download-429243.html • For JDK 7, download the files here: http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html • For JDK 8, update 150 and earlier, download the files here: http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html • For JDK 8, update 151 and later, and JDK 9, edit the file `conf/security/java.security` and uncomment the line `crypto.policy=unlimited`.

Related Information

Download and Install SAP Common Crypto Library
JDBC Connection Properties [page 441]
LDAP User Authentication

3.3 HTTP Proxy Client Connections

An HTTP proxy is a server that only authenticates HTTP connection requests.

Using direct Transmission Control Protocol (TCP) connections is recommended for external servers. If your network does not allow direct TCP connections, then you must connect via a proxy.

When connecting through an HTTP proxy for SAP HANA Cloud, specify the `proxyHttp` connection property (JDBC and ODBC) or the *-proxyhttp* option (HDBSQL). This setting ensures that TCP traffic is forwarded via an `HTTP CONNECT` command through the proxy.

Related Information

JDBC Connection Properties [page 441]
ODBC Connection Properties [page 621]
SAP HANA HDBSQL Options [page 858]

3.3.1 Implement HTTP Proxy Client Connections

An HTTP proxy connection allows TCP connections to be connected via an HTTP proxy for JDBC- and SQLDBC-based clients.

Context

Use HTTP proxy connections when direct TCP connections to an external server are not possible.

Procedure

1. Specify the `proxyHostName` and `proxyPort` connection properties.
2. (Optional) Specify the `proxyUserName` and `proxyPassword` connection properties if your HTTP proxy requires user authentication.
3. Set the `proxyHttp` connection property to TRUE. Do not specify the `webSocketURL` connection property.

Example

Use JDBC to connect to SAP HANA Cloud via an HTTP proxy:

```
jdbc:sap://724b1417-d273-4115-a1f0-7a857742eefe.hana.canary-
eu10.hanacloud.ondemand.com:443/?
proxyHTTP=true&proxyHostName=httpproxyhost.company.com&proxyPort=8080
```

Use ODBC to connect to SAP HANA Cloud via and HTTP proxy:

```
Driver={HDBODBC};uid={username};pwd={password};serverNode=724b1417-d273-4115-
a1f0-7a857742eefe.hana.canary-
eu10.hanacloud.ondemand.com:443;proxyHTTP=true;proxyHostname=httpproxy.company.co
m;proxyPort=8080
```

Related Information

JDBC Connection Properties [page 441]
ODBC Connection Properties [page 621]
SAP HANA HDBSQL Options [page 858]

3.4 Mutual Authentication (SAP HANA Platform)

Mutual authentication provides an additional level of security in a client server environment.

In typical transport layer security (TLS) communication, the client only authenticates the server to verify that it is communicating with the right server and to prevent man-in-the-middle attacks (when an intermediary alters the communication between the client and the server).

Configuring your system with mutual authentication allows the server to validate a certificate provided by the client as an additional authentication mechanism.

3.4.1 Implement Mutual Authentication (SAP HANA Platform)

SAP HANA allows you to configure secure client-server communication with mutual authentication for SAP HANA Platform. Mutual Authentication is a security scheme where each client has its own key pair and public certificate, which is validated by the SAP HANA server.

Context

SAP HANA requires a restart as part of setting up mutual authentication.

Once enabled, client authentication is enforced for all SQL connections to your instance via the SQL port.

Procedure

1. Create a key pair and public certificate for the client. For security reasons, each client should have its own key pair and public certificate.

Create a certificate by using one of the methods below, depending on the type of certificate required by your site:

Certificate Type	Steps

A certificate signed by a certificate authority (CA)	1. Use `sapgenpse` (located in the `/usr/sap/<instance_id>/HDB00/exe` directory on UNIX, and in the `\hdbclient` directory on the client) to generate a key pair and a certificate request.
	The following example creates a personal security environment (PSE) file named `sapcli.pse` and a request file named `sapcli.req`:
	`sapgenpse gen_pse -p sapcli.pse -r sapcli.req`
	2. Select *Enter* twice to not use a PIN. The client does not support PIN-protected PSE files.
	3. Enter the distinguished name for the certificate (the server ignores distinguished names for client certificates). Distinguished names use a format similar to this: `CN=Client 001`
	4. Send this file to a certificate authority (who signs it with their own private key). The `sapcli.req` file contains the certificate request.
	Note
	Although `ca.srl` is not included in the command, OpenSSL looks for this file as part of the operation and reports an error if it cannot find `ca.srl` in the local directory.
	5. Ensure that `sapcli.crt` includes the certificate authority's certificate. Execute this command to append it to `sapcli.crt` if it is not included:
	`cat ca.crt >> sapcli.crt`
	6. Invoke `sapgenpse` again to add this information to the PSE file once the client has received its signed certificate from the certificate authority:
	`sapgenpse import_own_cert -p sapcli.pse -c sapcli.crt`
	The file received from the certificate authority containing the client's signed certificate must also include the certificate authority's public certificate.

A self-signed certificate	1. Use `sapgenpse` to generate a key pair and a certificate (this example creates a PSE file named `sapcli.pse`): `sapgenpse gen_pse -p sapcli.pse` 2. Select *Enter* twice to not use a PIN. The client does not support PIN-protected PSE files. 3. Enter the distinguished name for the certificate (the server ignores distinguished names for client certificates). Distinguished names use a format similar to this: `CN=Client 001`

Certificate Type	Steps
Importing an existing certificate	Use `sapgenpse` to import an existing self-signed certificate or a certificate signed by a certificate authority (you cannot use the `-r` option when importing a self-signed certificate): `sapgenpse import_p8 -p sapcli.pse -c sapcli.crt -r ca.crt sapcli.key` • `sapcli.crt` – contains the client's public certificate and must be in PEM format. • `sapcli.key` – contains the client's private key in PKCS8 format. • `ca.crt` – is the certificate authority's public certificate in PEM format.

2. (Optional) Use the `sapgenpse get_my_name` parameter to verify that `sapcli.pse` contains the expected certificate:

```
sapgenpse get_my_name -p sapcli.pse
```

3. Configure the server for mutual authentication. For the server to trust the client, the server must have a copy of the public certificate of the certificate authority that signed the client's certificate.
   a. Use `sapgenpse` to extract the public certificate from the client's PSE file (these examples extract the certificate from `sapcli.pse` and write it to a file named `client_root.crt`):
      - Client certificates are self-signed – run:

```
sapgenpse export_own_cert -p sapcli.pse -o client_root.crt
```

      - Client certificates are not self-signed – run:

```
sapgenpse export_own_cert -r -p sapcli.pse -o client_root.crt
```

   b. Issue the following command to add the root certificate to the server's PSE file:

```
sapgenpse maintain_pk -p sapsrv.pse -a client_root.crt
```

      The server expects `sapsrv.pse` to be located in the `$SECUDIR` directory. Specify a PSE file with a different name or path with the `sslKeyStore` and `sslTrustStore` server configuration properties.

      > **Note**
      >
      > You can also perform this step by using an in-memory store (that is, using in-database certificate management rather than file-based certificate management).

   c. Restart SAP HANA.
   d. Set the `sslValidateCertificate` server communication property to "true" to enable mutual authentication by connecting to SYSTEMDB and executing the following statement:

```
ALTER SYSTEM ALTER CONFIGURATION ('global.ini','SYSTEM') SET
('communication','sslValidateCertificate') = 'true' WITH RECONFIGURE;
```

   e. (Optional) To force all connections to use encryption and mutual authentication, set the `sslEnforce` server communication property to "true" by connecting to SYSTEMDB and executing the following statement:

```
ALTER SYSTEM ALTER CONFIGURATION ('global.ini','SYSTEM') SET
('communication','sslEnforce') = 'true' WITH RECONFIGURE;
```

> **Note**
>
> Communication properties are indicated in the default configuration change blocklist (located in `multidb.ini`), meaning they cannot initially be changed in tenant databases. Instead, you must change them while in the system database. You can remove these properties from the change blocklist if this is appropriate for your site.

4. Configure the client to trust the server certificate. When using mutual authentication, clients must verify that the server's certificate is trusted by adding the public certificate of the certificate authority that signed the server's certificate into the client PSE file (these examples extract the certificate from `sapsrv.pse` and write it to a file named `server_root.crt`).
   a. Use `sapgenpse` to export the certificate.

      If the server certificates are self-signed, run:

      ```
      sapgenpse export_own_cert -p sapsrv.pse -o server_root.crt
      ```

      If the server certificates are not self-signed, run:

      ```
      sapgenpse export_own_cert -r -p sapsrv.pse -o server_root.crt
      ```

   b. Add the root certificate, `server_root.crt`, to the client's PSE file:

      ```
      sapgenpse maintain_pk -p sapcli.pse -a server_root.crt
      ```

   c. (Optional) Verify that the previous step completed successfully by viewing a list of all the certificate authorities the client trusts:

      ```
      sapgenpse maintain_pk -l -p sapcli.pse
      ```

5. Provide the client certificate and key pair to the client driver so it can connect to SAP HANA. How you provide the client PSE depends on which `crypto` provider you are using:

Crypto Provider	Description
Common Crypto Library	The Common Crypto Library provider uses the client PSE file directly. By default, the driver expects the file name to be `sapcli.pse` and to be in the `$SECUDIR` (%SECUDIR% for Microsoft Windows) directory. You can specify a PSE file with a different name or path with the `sslKeyStore` and `sslTrustStore` client configuration properties. This example uses `hdbsql` to connect through the Common Crypto Library provider to an SAP HANA server configured with mutual authentication: `hdbsql -n myhost.com:30015 -u MyUser -p MyPassword123 -e`

Crypto Provider	Description

OpenSSL	The OpenSSL provider cannot use PSE files directly; it expects the client key pair and public certificate to be in PEM and PKCS8 format.
	1. Export the PSE key to a PKCS8 format by using the `sapgenpse export_p8` parameter (this example exports the `sapcli.pse` key to the `sapcli.pem` file):
	`sapgenpse export_p8 -p sapcli.pse sapcli.pem`
	2. Set the following connection properties:
	• The `sslCryptoProvider` client connection parameter to `openssl` so it uses the OpenSSL provider instead of the Common Crypto Library provider.
	• `sslTrustStore` to the file containing the server's root certificate authority that was created in step 4 (`server_root.crt` in the example above).
	• `sslKeyStore` to the file created in the previous step (`sapcli.pem` in the example above).
	This example uses SAP HANA HDBSQL to connect to an SAP HANA server with mutual authentication using the OpenSSL provider:
	`hdbsql -n myhost.com:30015 -u MyUser -p MyPassword123 -e -Z sslCryptoProvider=openssl -Z sslTrustStore=server_root.crt -Z sslKeyStore=sapcli.pem`

Java	The JDBC driver uses the TLS support that is included with the JDK. Supply the client key pair and public certificate to the JDK by using the JDBC driver in PKCS12 format:
	1. Use `sapgenpse` to export the keys to the PKCS12 format (this example exports the keys to a file named `sapcli.p12`):
	`sapgenpse export_p12 -z mypassword -p sapcli.pse sapcli.p12`
	2. Use the Java `keytool` command to convert the server's root certificate to the Java `keystore` file format:
	`keytool -importcert -keystore server_root.jks -file server_root.crt -storepass mypassword`
	3. Connect to an SAP HANA server with mutual authentication. You must supply:
	• The `keyStoreType`, `keyStore`, and `keyStorePassword` connection properties to provide the client key pair and public certificate to the JDBC driver.
	• The `trustStoreType`, `trustStore`, and `trustStorePassword` connection properties to provide the root certificate that signed the server's certificate (unless the server's certificate was signed by a certificate authority that is already included in the default Java keystore).
	This example uses JDBC to connect to an SAP HANA server with mutual authentication:
	`java -jar ngdbc.jar -u MyUser,MyPassword123 -n myhost.com:30015 -o encrypt=true -o keyStoreType=pkcs12 -o keyStore=sapcli.p12 -o keyStorePassword=mypassword -o trustStoreType=jks -o trustStore=server_root.jks -o trustStorePassword=mypassword -c "SELECT * FROM MYTABLE"`

Related Information

SAP HANA Platform: Server-Side TLS/SSL Configuration Properties for Internal Communication
SAP HANA Cockpit: Certificate Collections
SAP HANA Platform: Export a Client Certificate
SAP HANA Platform: TLS/SSL Configuration on the Client
SAP HANA Platform: Secure Communication Between SAP HANA and JDBC/ODBC Clients

4 SAP HANA Application Programming Interfaces

This section lists the available interfaces that you can use to connect the client to an SAP HANA database.

4.1 ADO.NET Application Programming

ADO.NET is the data access services component of Microsoft .NET Framework and .NET Core application development. It allows you to access data in relational database systems.

The ADO.NET object model is an all-purpose data access model. ADO.NET components are designed to factor data access from data manipulation. Two central components of ADO.NET do this: the `DataSet` and the data provider, which is a set of components including the `Connection`, `Command`, `DataReader`, and `DataAdapter` objects.

The .NET Core provider runs on Linux, macOS, and Microsoft Windows.

How to Create ADO.NET Applications in Visual Studio

Microsoft Visual Studio 2015 and earlier	Use the SAP HANA Data Provider for ADO.NET. The SAP HANA Data Provider for ADO.NET implements the Sap.Data.Hana namespace and allows you to write programs in any .NET supported language, such as Microsoft C# or Microsoft Visual Basic, and access data from SAP HANA database servers. For a list of supported Microsoft .NET Framework versions, see 3165810.

Related Information

SAP Note 3136015

4.1.1 The SAP HANA Provider for ADO.NET

The SAP HANA Provider for ADO.NET enables .NET applications to connect to SAP HANA databases.

The provider is supported on both .NET technology platforms:

- The Microsoft .NET Framework (3.5 and 4.5), on Microsoft Windows 32-bit and 64-bit
- .NET Core (2.1, compatible with 2.2 and 3.x) on Microsoft Windows 32-bit and 64-bit, Linux x86_64, Mac OS X

Related Information

The .NET Core Provider [page 74]
The Microsoft .NET Framework Provider [page 75]

4.1.1.1 ADO.NET Connection Properties

When connecting to an SAP HANA database using ADO.NET, you can specify ADO.NET connection properties or SQLDBC connection properties.

For information regarding SAP HANA client and server cross-version compatibility, see https://launchpad.support.sap.com/#/notes/1906576

Specify a connection string in a .NET application when the connection object is created or by setting the ConnectionString property of a connection object.

```
HanaConnection conn = new HanaConnection("<connection-string>");
```

```
HanaConnection conn = new HanaConnection();
conn.ConnectionString = "<connection-string>";
```

Specify connection properties as keyword=value pairs, separated by semicolons. For example:

```
HanaConnection conn = new HanaConnection(
        "server=hana-server:30015;userid=JSmith;password=secret");
```

Connection property names are case insensitive. For example, `UserID` and `userid` are equivalent. If a connection property name contains spaces, then they must be preserved.

Connection property values can be case sensitive. For example, passwords are usually case sensitive.

Connection Properties

Property	Value	Default	Description
`connection lifetime`	timeout in seconds	0	Specifies the maximum lifetime for a connection that is to be pooled. If a connection has been open for longer than the maximum lifetime, then the connection is not pooled when it is closed. The default is 0, which means no maximum.
`connection timeout`	timeout in seconds	15	Specifies the length of time (in seconds) to wait for a connection to the SAP HANA database server before terminating the attempt and generating an error. The default is 15. The alternate form is `CONNECT TIMEOUT`.
`enlist`	boolean	TRUE	Specifies whether the connection should be automatically enlisted in the creation thread's current transaction context.
`initString`	`<string>`	Empty string	Specifies a SQL statement that is executed immediately after a connection is established to the SAP HANA database server.
`max pool size`	`<number>`	100	Specifies the maximum size of the connection pool.
`min pool size`	`<number>`	0	Specifies the minimum size of the connection pool.
`password`			Specifies the SAP HANA database server user password. The alternate form is `PWD`.

Property	Value	Default	Description
`persist security info`	boolean	FALSE	Indicates whether the `password` (`PWD`) connection property must be retained in the `ConnectionString` property of the connection object. When set to TRUE, the application can obtain the user's password from the `ConnectionString` property if the `password` (`PWD`) connection property was specified in the original connection string.
`pooling`	boolean	TRUE	Enables or disables connection pooling.
`PWD`	`<password>`	N/A	Specifies the user password. This property is optional depending on the used authentication method.
`server` | `serverNode`	`<host>:<port>`[, `<host>:<port>`]	N/A	Specifies the SAP HANA server address or addresses. If your system contains a failover standby node, then you can specify it as an additional optional address.
`UID`	`<user>`	N/A	Specifies the user name. This property is optional depending on used authentication method. The alternate forms are `Username` and `UserID`.

SQLDBC Connection Properties

The following table lists SQLDBC connection properties (property names are case insensitive) that can also be used with ADO.NET:

> **Note**
>
> A valid setting for a Boolean property can be: “1”, “TRUE”, “YES”, or “ON” for a TRUE setting or “0”, “FALSE”, “NO”, or “OFF” for a FALSE setting. The setting of a Boolean property is case-insensitive.

Property	Value	Default	Description
`abapVarCharMode`	Boolean	FALSE	Specifies that in ABAP VARCHAR mode, a single space and an empty string are considered the same.
`allowLocalCompress`	Boolean	TRUE	Determines whether or not connections from the same machine as the SAP HANA server are compressed. The `compress` parameter and/or the `indexserver.ini` > *session* > *compression* server configuration control whether or not network compression is enabled. **Note** If the connection uses a proxy, do not use `allowLocalCompress`=FALSE as the connection may not be compressed when required.
`allowReconnectOnSelect`	Boolean	OFF	Allows reconnects to happen for SELECT queries which fail during a receive. By default, this property is set to OFF, which allows for backward compatibility.

Property	Value	Default	Description
`bindAddress`	`<ip-address>`	NULL	Specifies that the provided address is used as the source address for the socket connection. By default, the address preferred by the OS is used. If setting `bindAddress` to a hostname, that hostname should only resolve to a single IP address. In the majority of network configurations, specifying `bindAddress` is not required. **Note** Only specify `bindAddress` if it is required to connect in your network configuration.
`chopBlanks`	Boolean	FALSE	Removes trailing blanks from character output.
`chopBlanksInput`	Boolean	FALSE	Removes trailing blanks from character input.
`communicationTimeout`	Timeout in milliseconds	0 (disabled)	Aborts communication attempts to the server after the specified timeout. If set to a non-zero value, the driver will wait for a maximum of the specified number of milliseconds for any request to the server to complete. If the time has expired and a response has not been received, the driver returns an error.

Property	Value	Default	Description
`compress`	Boolean	FALSE	Enables or disables network compression for the connection. The `indexserver.ini` > *session* > *compression* server configuration defaults to FALSE. Setting this parameter overrides the setting of the `indexserver.ini` > *session* > *compression* parameter. The `allowLocalCompress` property controls if clients running on the same machine as the server use network compression when compression is requested.
`connDownRollbackError`	Boolean	FALSE	Setting this property to TRUE specifies that if the database connection is down due to network or SAP HANA server problems, the client rolls back the last transaction and sends and error to the application. The application can resend the last transaction to the server after properly handling the error. The following conditions must be true: • The session is recoverable in the server • There is a written transaction (uncommitted operation) in the session • The client failed to send the last SQL statement, or the last statement was sent to the server successfully, but both auto-commit and DDL auto-commit in the client are off. By default, this property is set to FALSE.

Property	Value	Default	Description
`connectTimeout`	Timeout in milliseconds	0 (limited by the `nodeConnectTimeout` connection property)	Aborts connection attempts after the specified timeout. `connectTimeout` limits the overall time for the entire connection to try all server addresses specified. See `nodeConnectTimeout` for per-node timeout.
`currentSchema`	`<schema>`	CURRENTUSER	Sets the current schema, which is used for identifiers without a schema.
`cursorHoldabilityType`	NONE, COMMIT, ROLLBACK, COMMIT_AND_ROLLBACK	COMMIT	Specifies whether the cursor for a result set remains open after the transaction is committed or rolled back. If a cursor remains open, then the application is able to fetch more rows from it. NONE means the cursor is always closed after a commit or rollback, while COMMIT means the cursor remains valid after a commit, but not after rollback. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources. Support for holding a cursor over ROLLBACK or COMMIT_AND_ROLLBACK is available in SAP HANA servers beginning with SAP HANA 2.4.

Property	Value	Default	Description
`emptyTimestampIsNull`	Boolean	TRUE	When enabled, DATE, TIME, SECONDDATE, and TIMESTAMP values inserted as empty strings are returned as NULLs. When disabled, these values are returned as the following out-of-band values: • `DATE` - one day before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 00:00.000000000 UTC BC • `TIME` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `SECONDDATE` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `TIMESTAMP` - one hundred nanoseconds before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.999999900 UTC BC

Property	Value	Default	Description
`ignoreTopology`	Boolean	FALSE	Controls how the client uses the topology returned from the server. TRUE indicates to use the topology unless port-forwarding is detected and FALSE indicates to always ignore the topology. It is an error to specify both ignoreTopology=TRUE and siteType=PRIMARY or SECONDARY.
`isolationLevel`	TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE	TRANSACTION_READ_COMMITTED	Sets the isolation level for the connection.
`isoTimestampOutput`	`<string>`	FALSE	Displays whether or not 'T' is used as the separator between the Date and Time parts of a timestamp retrieved as a string: TRUE/FALSE. If FALSE, then a space character is used as the separator.
`key`	`<username>`	Empty	Specifies the key for the hdbuserstore.
`maxLazyDroppedStatements`	`<number>`	50	Controls the maximum number of queued statements to be dropped. Setting this value to 0 disables the lazy drop feature and causes a request to be sent to the server each time a statement is dropped by the application.
`maxVersion`	`<string>`		Specifies the maximum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.

Property	Value	Default	Description
`minVersion`	`<string>`		Specifies the minimum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.
`nodeConnectTimeout`	Timeout in milliseconds	60000	Aborts the connection attempt after the specified maximum timeout. The connection attempt typically fails immediately if the server cannot be found. If connection attempts to a particular server address are slow to fail, then using this connection property reduces the time before a successful connection by minimizing the time waiting to connect to individual addresses. On a host with multiple resolved IP addresses, the network timeout for each attempt to connect to an IP address will be equal to the `nodeConnectTimeout` divided by the number of resolved IP addresses. If this option is set to zero, the `connectTimeout` setting will be used. If both of the `nodeConnectTimeout` and `connectTimeout` are set to zero, then SQLDBC will use the system TCP/IP socket connect timeout to connect to the IP address.

Property	Value	Default	Description
`packetCaching`	Boolean	TRUE	Caching packets reduces the need to continually allocate and deallocate memory for packets. However, this increases the peak memory usage of the application, and for how long it stays at or near that peak.
`packetSize`	`<number>` in bytes	1048576 (1 MB)	Sets the maximum size of a request packet sent from the client to the server, in bytes. In some cases, request packets can be larger (for example, if a SQL statement or parameter value is larger than the packetSize). The minimum is 1 MB.
`packetSizeLimit`	`<number>` in bytes	2147483647 (2 GB)	Sets the maximum size for a request or reply packet, in bytes. If SQLDBC needs to allocate a packet larger than the limit, then it returns an error. If a reply packet is too large, then the connection is dropped. The minimum value is the packetSize value; the maximum value is 2 GB.

Property	Value	Default	Description
`prefetch`	Boolean	TRUE	Controls result set prefetching. When enabled, result set prefetch requests the next block of rows of a result set while the application is still processing the current block of rows. When fetching large result sets, result set prefetch can improve performance; however, it can also result in the client using more memory. In some cases, the server JobExecutor thread could be blocked from sending the reply for an indefinite period of time until the client receives the prefetch reply. If the result set contains LOB columns, then result set prefetch is always disabled.
`{ proxyHostname | proxy_host }`	`<host-name>`		Specifies the host name of the proxy server. The default proxy server is SOCKS5. If `proxyHttp` is set to true, the proxy server is assumed to be an HTTP proxy.
`proxyHttp`	Boolean	FALSE	When set to TRUE, enables HTTP proxy authentication and implies that ENCRYPT=TRUE. This property is only applicable if `proxyHostname` is specified and `webSocketURL` is not specified.

Property	Value	Default	Description
`{ proxyPassword | proxy_password }`	`<password>`		Specifies the password for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the Cloud Connector location ID value. The value must be entered as a base64 encoded value with a "1." prepended to the string.
`{ proxyPort | proxy_port }`	`<port-number>`		Specifies the port of the proxy server. The default is 1080.
`{ proxyScpAccount | proxy_scp_account }`	SAP BTP, Cloud Foundry Environment: `<location-ID>` SAP BTP, Neo Environment: `<subaccount>.<location-ID>`		Provides SAP Cloud Connector routing information to the Cloud Connector SOCKS proxy. In an SAP BTP, Cloud Foundry Environment, provide an optional `<locationID>` in plaintext. Internally, the value is converted to base64 encoding. This value overrides the `proxyPassword` connection property. In an SAP BTP, Neo Environment, either `proxyScpAccount` or `proxyUsername` must be specified. `<subaccount>` is mandatory and `<locationID>` is optional. Both values are entered as plaintext strings and are internally converted to base64 encoding. This value overrides the `proxyUsername` connection property.

Property	Value	Default	Description
`{ proxyUserName | proxy_userid }`	`<user-name>`		Specifies the user name for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the JSON Web Token (JWT) value. When using an SAP HANA Cloud Connector in the SAP BTP, Neo Environment, this field is used to provide the subaccount and location ID details. The values must be base64 encoded manually using the format `1.base64(subaccount).base64(locationID)`.
`reconnect`	Boolean	TRUE	Controls whether the system automatically reconnects to the database instance after a command timeout or when the connection is broken. Reconnecting restores the old state (for example, if no transaction was open).

Property	Value	Default	Description
`resolveHostName`	IPv4First IPv6First	IPv4First	Controls which addresses to use in a server address, especially if an address or hostname resolves to multiple IP addresses. In all cases, stop attempting further resolved addresses once a TCP/IP connection is successfully established. The following values can be used with `resolveHostName`: • IPv4First (default) - try all resolved addresses starting with IPv4 addresses. • IPv6First - try all resolved addresses starting with IPv6 addresses.
`sessionVariable:<key-name>`	`<session variable value>`	None except for APPLICATION and APPLICATIONUSER keys	Sets the session variable `<key-name>` to `<session variable value>`. For example, to set the session variable with key `APPLICATION`, use `sessionVariable:APPLICATION=MyApplication`.

Property	Value	Default	Description
`statementCacheSize`	`<number>`	0 (no statements cached)	Controls the maximum number of cached prepared statements. Caching SQL statements that are frequently prepared and dropped can improve performance. DDL can cause behavior changes when statement caching is enabled. **Note** If it is possible that the definition of objects referred to by the prepared statements may change because DDL is executed while you are connected, then it is recommended that you do not enable prepared statement caching.
`statementRoutingFailureBackoff`	Boolean	TRUE	Enables or disables the re-establishment of an internal connection to a specific host for a time period, after the internal connection fails during the preparation or execution of a statement routed to that host. The minimum time period is 1 minute. The time period is doubled after each subsequent internal connection establishment or usage failure for the same node, up to a maximum of 1 hour. Every successful internal connection re-establishment and usage resets the time period to the minimum value.

Property	Value	Default	Description
`statementRoutingWarnings`	Boolean	FALSE	Returns a SQL warning to the application with SQLCODE -11001 or -11003 that gives information about the routing failure of a prepared statement.
`tcpKeepAliveCount`	`<number-of-retries>` This value is ignored if it is 0.	5	Controls the number of KeepAlive packets sent to the server before the driver assumes that the connection is closed. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.
`tcpKeepAliveIdle`	`<seconds>` This value is ignored if it is 0.	200	Controls the number of seconds that the connection is idle before the driver sends a KeepAlive packet. If 0 is specified, this value is determined by the operating system.
`tcpKeepAliveInterval`	`<seconds>` This value is ignored if it is 0.	4	Controls the number of specified seconds before the driver sends an additional KeepAlive packet if there was no response to the previous KeepAlive packet. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.

Property	Value	Default	Description
`timestampPadding`	Boolean	TRUE	In a timestamp where the format contains a fraction, this property determines whether the fractional portion of the timestamp will be 7 or 9 digits. When disabled, the fractional portion is 7 digits to match how the server displays timestamps. Otherwise, it is 9 digits. **Note** ABAP timestamps are not affected regardless of the value of `timestampPadding`.
`traceFile`			Specifies the trace file name. Only the stdout and stderr values are supported.
`traceOptions`			Implicitly enables tracing for all SQLDBC-based client connections. See the HDB_SQLDBC_TRACEOPTS description in the **Environment Variables** topic for more information.
`webSocketURL` (SAP HANA Service only)	`<webSocketUrl>`		Specifies the WebSocket URL used to connect to a SAP HANA server. The connection property value cannot be empty. WebSockets connections have the following default connection property settings: • encrypt=TRUE • compress=TRUE • reconnect=FALSE

Security Connection Properties

Property	Value	Default	Description
`{ cseKeyStorePassword | clientside_encryption_keystore_password }`	`<string>`		Provides the password for the local key store. This option is required when using client-side encryption. Passwords must be at least eight characters and have at least one upper case character, one lower case character, and one number.
`authenticationMethods`	`<comma-separated-string>`		Provides a comma-separated list of authentication methods to be used by the client. Possible values are: password, pbkdf2, ldap, saml, jwt, saplogon, kerberos, sessioncookie, and X509.
`authenticationX509`	`<string>`	`<value-of-sslKeyStore>`	Provides a string containing the X509 certificate or the name of a file containing the X509 certificate. The certificate must be PEM-encoded in PKCS8 format. Defaults to the existing value of `sslKeyStore`, if specified.
`authenticationX509Password`	`<password>`		With CommonCryptoLib, specifies the PIN for the PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`encrypt`	Boolean	FALSE	Enables or disables TLS encryption. For SAP HANA versions 2.5 and lower, set encrypt=TRUE when connecting to port 443 (SAP HANA Cloud). For SAP HANA versions 2.6 and higher, encrypt=TRUE is not required when connecting to port 443.

Property	Value	Default	Description
`sslCryptoProvider`	{ commoncrypto | openssl | mscrypto }	1. commoncrypto (if installed) 2. openssl/mscrypto	Specifies the cryptographic library provider used for TLS communication. If you specify a value for this property, then you must also explicitly specify paths in both the `sslKeyStore` and `sslTrustStore` properties to avoid configuration issues. If CommonCryptoLib is not available, OpenSSL is used by default in Linux environments, and msCrypto in Microsoft Windows environments. OpenSSL is not available on Microsoft Windows. **Note** Check the `client` folder of the installation package to see if `COMMONCRYPTOLIB.TGZ` is present. If not, you can download CommonCryptoLib separately. For instructions on how to download CommonCryptoLib, see "Download and Install SAP Common Crypto Library" in the *SAP HANA Client Installation and Update Guide*.
`sslHostNameInCertificate`	`<string>`	Empty	Specifies the host name used to verify server's identity. The host name specified here verifies the identity of the server instead of the host name with which the connection was established. For example, in a single-host system, if a connection is established from a client on the same host as the server, then a mismatch would arise between the host named in the certificate (actual host name) and the host used to establish the connection (localhost). If you specify `*` as the host name, then the server's host name is not validated. Other wildcards are not permitted.

Property	Value	Default	Description
`sslKeyStore`	`<file>` | `<PEM-encoded-identity>` msCrypto: MY | Root | Trust | CA	`$SECUDIR/sapcli.pse` (CommonCryptoLib) `$HOME/.ssl/key.pem` (OpenSSL) MY (msCrypto)	Specifies the path to the keystore file that contains the client's identity. An identity is used for mutual TLS authentication and consists of the client's private key, the client's certificate, and, optionally, the certificate of the signing authority that signed the client's certificate. If you are using CommonCryptoLib, then the PSE file must also contain the trust store (for example, the server's public certificates) and the sslTrustStore property should be empty. If you are using CommonCryptoLib, but not doing mutual TLS authentication, then the PSE file contains only the trust store. If you are using CommonCryptoLib, then use the SAPGENPSE tool (installed with CommonCryptoLib) to create the client PSE (`sapcli.pse`) and generate the client's keys. You must also import the server's public certificates into `sapcli.pse`. Typically, this is the root certificate or the certificate of the certification authority that signed the server's public certificates. See *SAP Note 1718944 (SAP HANA DB: Securing External SQL Communication (CommonCryptoLib))*. If you are using OpenSSL, use OpenSSL tools to create the required keystore file. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.

Property	Value	Default	Description
`sslMinProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	TLS12	Controls the minimum TLS version. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The lowest protocol version is accepted in TLS handshakes.
`sslMaxProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	MAX	Controls the maximum TLS version. MAX allows this parameter to be future-proof in the event that new TLS versions are supported. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The highest negotiated protocol version is used in TLS handshakes.
`sslKeyStorePassword`	`<password>`		With CommonCryptoLib, specifies the PIN for the keystore PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`sslSNIHostname`	`<string>`	Empty	Specifies the name of the host that is attempting to connect at the start of the TLS handshaking process.
`sslSNIRequest`	Boolean	TRUE	Specifies whether SNI requests are enabled for TLS connections: TRUE/FALSE.

Property	Value	Default	Description
`sslTrustStore`	`<file>` | `<PEM-encoded-trust-store>` msCrypto: MY | Root | Trust | CA	`$HOME/.ssl/trust.pem` (OpenSSL) sapcli.pse (CommonCryptoLib) MY (msCrypto)	Specifies the path to a trust store file that contains the server's public certificates if using OpenSSL. Typically, the trust store contains the root certificate or the certificate of the certification authority that signed the server's public certificates. If you are using the cryptographic library CommonCryptoLib or msCrypto, leave this property empty. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.
`sslUseDefaultTrustStore`	Boolean	TRUE	(Only applies to the OpenSSL Crypto Provider) Set to FALSE in order to disable the use of the default Certificate Authorities provided with OpenSSL or the Operating System.
`sslValidateCertificate`	Boolean	TRUE	Specifies whether to validate the server's certificate.

Connection String Examples

- Connect to an SAP HANA database server and then execute a SET SCHEMA DEMO statement to set the current schema.

```
HanaConnection conn = new HanaConnection(
    "Server=hana-db:30015;UserID=JSmith;password=secret;" +
    "Current Schema=DEMO");
```

- Connect to an SAP HANA database server by using a user ID and password that were provided by the user.

```
HanaConnection conn = new HanaConnection();
conn.ConnectionString = "Server=hana-db:30015;Database=GA1;" +
    "UID=" + user_name + ";PWD=" + pass_code;
```

4.1.1.2 Connection Examples for the SAP HANA Data Provider for ADO.NET

To connect to an SAP HANA database server by using the SAP HANA Data Provider for ADO.NET, create a `HanaConnection` object.

Specify the connection string when creating the object or establish it later by setting the `ConnectionString` property.

A well-designed application handles any errors that occur when it attempts to connect to an SAP HANA database server.

A connection to the SAP HANA database server is created when the connection is opened, and it is released (pooled) when the connection is closed.

Microsoft C# HanaConnection Example

The following Microsoft C# code creates a button click handler that opens a connection to an SAP HANA database server and then closes it. It includes an exception handler.

```
private void button1_Click(object sender, EventArgs e)
{
    HanaConnection conn = new HanaConnection(
        "server=hana-db:30015;userid=JSmith;password=secret");
    try
    {
        conn.Open();
        conn.Close();
    }
    catch (HanaException ex)
    {
        MessageBox.Show(ex.Errors[0].Source + " : " +
             ex.Errors[0].Message + " (" +
             ex.Errors[0].NativeError.ToString() + ")",
             "Failed to connect");
    }
}
```

Microsoft Visual Basic HanaConnection Example

The following Microsoft Visual Basic code creates a button click handler that opens a connection to an SAP HANA database server and then closes it. It includes an exception handler.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New HanaConnection( _
        "server=hana-db:30015;userid=JSmith;password=secret")
    Try
        conn.Open()
        conn.Close()
    Catch ex As HanaException
        MessageBox.Show(ex.Errors(0).Source & " : " & _
```

```
                ex.Errors(0).Message & " (" & _
                ex.Errors(0).NativeError.ToString() & ")", _
                "Failed to connect")
    End Try
End Sub
```

4.1.1.3 Connection Pooling (SAP HANA Data Provider for ADO.NET)

The data provider supports native .NET connection pooling. Connection pooling allows your application to reuse existing connections by automatically saving the connection to a pool so it can be reused, rather than repeatedly creating a new connection to the SAP HANA database server.

Connection pooling is enabled and disabled by using the `pooling` connection parameter. Connection pooling is enabled by default.

The maximum pool size is set in your connection string by using the `max pool size` connection parameter. The minimum or initial pool size is set in your connection string by using the `min pool size` connection parameter. The default maximum pool size is 100, while the default minimum pool size is 0.

```
server=hana-db:30015;pooling=true;max pool size=50;min pool
size=5;userid=JSmith;password=secret
```

When your application first attempts to connect to the SAP HANA database server, it checks the pool for an existing connection that uses the same connection parameters you specified. If a matching connection is found, then that connection is used. Otherwise, a new connection is used. When you disconnect, the connection is returned to the pool so that it can be reused.

4.1.1.3.1 Improve Application Performance with Connection Pooling (SAP HANA Data Provider for ADO.NET)

Connection pooling may improve the performance of applications that make multiple, brief connections to the database server.

If connection pooling is enabled for a connection, when it is disconnected, the connection is automatically cached and may be reused when the application reconnects. When an application opens a new pooled connection, the SAP HANA data provider always searches the connection pool for an available connection with same connection string.

For connection pooling in .NET applications, use the .NET `pooling` connection parameter. The data provider cleans up and reinitializes pooled connections.

If the application process connects again and there are cached connections available for the same connection string, then cached connection is reused. Connections remain in the cached state for the time specified by the `connection lifetime` connection parameter (0 by default, which causes pooled connections to have the maximum connection timeout).

Connection pooling does not occur for non-standard database authentication, such as Kerberos logins. Only user ID and password authentication is supported.

To ensure that connection pooling is transparent to the application, a connection is disconnected if a failure occurs when caching a connection. If a failure occurs when attempting to reuse a cached connection, then the database server attempts to connect normally.

A connection is cached if it is disconnected and the maximum number of connections specified by the `max pool size` connection parameter has not been reached. The connection is reinitialized, and the cached connection continues to be connected to the database server even though the application has disconnected it. The cleanup and reinitialization of a connection includes rolling back all outstanding transactions.

4.1.1.4 Error Handling (SAP HANA Data Provider for ADO.NET)

Design your application to handle any errors that occur. The data provider creates a HanaException object and throws an exception whenever errors occur during execution, and stores information about the exception in the object.

Each `HanaException` object consists of a list of `HanaError` objects, and these error objects include the error message and code.

Errors are different from conflicts. Conflicts arise when changes are applied to a database table. Include a process in your application to compute correct values or to log conflicts when they arise.

Microsoft C# Error Handling Example

The following Microsoft C# code creates a button click handler that opens a connection to an SAP HANA database server. If the connection cannot be made, then the exception handler displays one or more messages.

```
private void button1_Click(object sender, EventArgs e)
{
    HanaConnection conn = new HanaConnection(
        "server=<hana-server:port>;UserID=<uid>;password=<pwd>");
    try
    {
        conn.Open();
    }
    catch (HanaException ex)
    {
        for (int i = 0; i < ex.Errors.Count; i++)
        {
            MessageBox.Show(ex.Errors[i].Source + " : " +
             ex.Errors[i].Message + " (" +
             ex.Errors[i].NativeError.ToString() + ")",
             "Failed to connect");
        }
    }
}
```

Microsoft Visual Basic Error Handling Example

The following Microsoft Visual Basic code creates a button click handler that opens a connection to an SAP HANA database server. If the connection cannot be made, then the exception handler displays one or more messages.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New HanaConnection( _
        "server=<hana-server:port>;UserID=<uid>;password=<pwd>")
    Try
        conn.Open()
    Catch ex As HanaException
        For i = 0 To ex.Errors.Count - 1
            MessageBox.Show(ex.Errors(i).Source & " : " & _
              ex.Errors(i).Message & " (" & _
              ex.Errors(i).NativeError.ToString() & ")", _
              "Failed to connect")
        Next i
    End Try
End Sub
```

4.1.1.5 Tracing Support (SAP HANA Data Provider for ADO.NET)

The data provider supports two tracing technologies. You can use the .NET tracing feature and you can use the HanaSqlTrace class.

.NET tracing support is described here. Refer to the HanaSqlTrace class for additional tracing capabilities.

By default, .NET tracing is disabled. To enable tracing, specify the trace source in your application's configuration file. The following is an example of a configuration file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
 <source name="Sap.Data.Hana"
         switchName="HanaSourceSwitch"
         switchType="System.Diagnostics.SourceSwitch">
  <listeners>
   <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
   <add name="EventListener"
        type="System.Diagnostics.EventLogTraceListener"
        initializeData="MyEventLog"/>
   <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
   <remove name="Default"/>
  </listeners>
 </source>
</sources>
<switches>
 <add name="HanaSourceSwitch" value="All"/>
 <add name="HanaTraceAllSwitch" value="1" />
 <add name="HanaTraceExceptionSwitch" value="1" />
```

```
 <add name="HanaTraceFunctionSwitch" value="1" />
 <add name="HanaTracePoolingSwitch" value="1" />
 <add name="HanaTracePropertySwitch" value="1" />
</switches>
</system.diagnostics>
</configuration>
```

There are four types of trace listeners referenced in this example.

`ConsoleTraceListener`	Tracing or debugging output is directed to either the standard output or the standard error stream. When using Microsoft Visual Studio, output appears in the *Output* window.
`DefaultTraceListener`	This listener is automatically added to the `Debug.Listeners` and `Trace.Listeners collections` by using the name **Default**. Tracing or debugging output is directed to either the standard output or the standard error stream. When using Microsoft Visual Studio, output appears in the *Output* window. To avoid duplication of output produced by the `ConsoleTraceListener`, the `DefaultTraceListener` listener is removed.
`EventLogTraceListener`	Tracing or debugging output is directed to an `EventLog` identified in the `initializeData` option. In the example, the event log is named **`MyEventLog`**. Writing to the system event log requires administrator privileges and is not a recommended method for debugging applications.
`TextWriterTraceListener`	Tracing or debugging output is directed to a `TextWriter` that writes the stream to the file identified in the `initializeData` option.

To disable tracing to any of the trace listeners described above, remove the corresponding `add` entry under `<listeners>`.

The trace configuration information is placed in the application's project folder in the `App.config` file. If the file does not exist, it can be created and added to the project by choosing *Add* » *New Item* and selecting *Application Configuration File*.

The `traceOutputOptions` can be specified for any listener and include:

`Callstack`	Write the call stack, which is represented by the return value of the `Environment.StackTrace` property.
`DateTime`	Write the date and time.
`LogicalOperationStack`	Write the logical operation stack, which is represented by the return value of the `CorrelationManager.LogicalOperationStack` property.
`None`	Do not write any elements.
`ProcessId`	Write the process identity, which is represented by the return value of the `Process.Id` property.
`ThreadId`	Write the thread identity, which is represented by the return value of the `Thread.ManagedThreadId` property for the current thread.

Timestamp Write the timestamp, which is represented by the return value of the `System.Diagnostics.Stopwatch.GetTimeStamp` method.

The example configuration file specifies trace output options for the `TextWriterTraceListener` only.

Limit what is traced by setting specific trace options. By default, the numeric-valued trace option settings are all 0. The trace options that can be set include:

HanaSourceSwitch `HanaSourceSwitch` can take any of the following values. If it is **`Off`**, then there is no tracing.

Off	Does not allow any events through.
Critical	Allows only critical events through.
Error	Allows critical and error events through.
Warning	Allows critical, error, and warning events through.
Information	Allows critical, error, warning, and information events through.
Verbose	Allows critical, error, warning, information, and verbose events through.
ActivityTracing	Allows the `Stop`, `Start`, `Suspend`, `Transfer`, and `Resume` events through.
All	Allows all events through.

Here is an example setting.

```
<add name="HanaSourceSwitch" value="Error"/>
```

HanaTraceAllSwitch All the trace options are enabled. You do not need to set any other options since they are all selected. You cannot disable individual options if you choose this option. For example, the following setting does not disable exception tracing.

```
<add name="HanaTraceAllSwitch" value="1" />
<add name="HanaTraceExceptionSwitch" value="0" />
```

HanaTraceExceptionSwitch All exceptions are logged. Trace messages have the following form.

```
<Type|ERR>
message='message_text'[ nativeError=error_number]
```

The nativeError=error_number text is only displayed if there is a `HanaException` object.

HanaTraceFunctionSwitch

All method scope entry/exits are logged. Trace messages have any of the following forms.

```
enter_nnn <class_name.method_name|API> [object_id#]
[parameter_names]
leave_nnn
```

The nnn is an integer representing the scope nesting level 1, 2, 3, ... The optional parameter_names is a list of parameter names separated by spaces.

HanaTracePoolingSwitch

All connection pooling is logged. Trace messages have any of the following forms:

```
<HanaConnectionPool.AllocateConnection|CPOOL>
connectionString='connection_text'
<HanaConnectionPool.RemoveConnection|CPOOL>
connectionString='connection_text'
<HanaConnectionPool.ReturnConnection|CPOOL>
connectionString='connection_text'
<HanaConnectionPool.ReuseConnection|CPOOL>
connectionString='connection_text'
```

HanaTracePropertySwitch

All property setting and retrieval is logged. Trace messages have any of the following forms.

```
<class_name.get_property_name|API> object_id#
<class_name.set_property_name|API> object_id#
```

4.1.1.5.1 Configure a Microsoft Windows Application for Tracing

Enable tracing on the TableViewer sample application by creating a configuration file that references the `ConsoleTraceListener` and `TextWriterTraceListener` listeners, removes the default listener, and enables all switches that would otherwise be set to 0.

Prerequisites

You must have Microsoft Visual Studio 2010 or a later version installed.

Procedure

1. Open the TableViewer project in Microsoft Visual Studio.

   Start Microsoft Visual Studio and open `Documents\SAP\Projects\TableViewer\TableViewer.sln`.

2. Create an application file named `App.config` and copy the following configuration setup:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
 <source name="Sap.Data.Hana"
         switchName="HanaSourceSwitch"
         switchType="System.Diagnostics.SourceSwitch">
  <listeners>
   <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
   <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
   <remove name="Default"/>
  </listeners>
 </source>
</sources>
<switches>
 <add name="HanaSourceSwitch" value="All"/>
 <add name="HanaTraceAllSwitch" value="1" />
 <add name="HanaTraceExceptionSwitch" value="1" />
 <add name="HanaTraceFunctionSwitch" value="1" />
 <add name="HanaTracePoolingSwitch" value="1" />
 <add name="HanaTracePropertySwitch" value="1" />
</switches>
</system.diagnostics>
</configuration>
```

3. Rebuild the application.
4. Click *Debug* » *Start Debugging*.

Results

When the application finishes execution, the trace output is recorded in the `bin\Debug\myTrace.log` file.

Next Steps

View the trace log in the *Output* window of Microsoft Visual Studio.

4.1.1.6 Transaction Processing (SAP HANA Data Provider for ADO.NET)

Use the `HanaTransaction` object to group statements together. Each transaction ends with a call to the `Commit` method, which either makes your changes to the SAP HANA database server permanent, or the `Rollback` method, which cancels all the operations in the transaction.

Once the transaction is complete, create a new `HanaTransaction` object to make further changes.

If you do not create a transaction, then the data provider operates in autocommit mode by default. There is an implicit COMMIT after each insert, update, or delete, and once an operation completes, the change is made to the SAP HANA database server. In this case, the changes cannot be rolled back.

Isolation Level Settings for Transactions

The READ COMMITTED isolation level is used by default for transactions. Specify the isolation level for a transaction by using the `IsolationLevel` property when you begin the transaction. The isolation level applies to all statements executed within the transaction. The data provider supports snapshot isolation.

The locks that the SAP HANA database server uses when you execute a SQL statement depend on the transaction's isolation level.

Microsoft C# HanaTransaction Example

This example shows how to wrap an UPDATE into a transaction so that it can be committed or rolled back. A transaction is created with a `HanaTransaction` object and linked to the execution of a SQL statement by using a `HanaCommand` object. The REPEATABLE READ isolation level (`RepeatableRead`) is specified so that other SAP HANA database server users cannot update the row. The lock on the row is released when the transaction is committed or rolled back. If you do not use a transaction, then the data provider operates in autocommit mode and you cannot roll back any changes that you make to a database table.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
string stmt = "UPDATE DEMO.Products SET UnitPrice = UnitPrice + 25.00 " +
    "WHERE ID >= 300 AND ID < 400";
bool goAhead = false;
HanaTransaction trans = conn.BeginTransaction(HanaIsolationLevel.RepeatableRead);
HanaCommand cmd = new HanaCommand(stmt, conn, trans);
int rowsAffected = cmd.ExecuteNonQuery();
if (goAhead)
    trans.Commit();
else
    trans.Rollback();
conn.Close();
```

4.1.1.7 Connection State (SAP HANA Data Provider for ADO.NET)

Once your application has established a connection to the SAP HANA database server, you can check the connection state to ensure that the connection is still open before sending a request to the SAP HANA database server.

If a connection is closed, then you can return an appropriate message to the user and/or attempt to reopen the connection.

The `HanaConnection` class has a `State` property that returns the state of the connection. Possible state values are `ConnectionState.Open` and `ConnectionState.Closed`.

The following code checks whether the `HanaConnection` object has been initialized, and if it has, it checks that the connection is open. The application returns a message to the user if the connection is not open.

```
if ( conn == null || conn.State != ConnectionState.Open )
{
    MessageBox.Show( "Connect to an SAP HANA database server first",
        "Not connected" );
    return;
}
```

4.1.1.8 Stored Procedures (SAP HANA Data Provider for ADO.NET)

The data provider supports calling SQL stored procedures.

Use the `ExecuteReader` method to call stored procedures that return result sets. Use the `ExecuteNonQuery` method to call stored procedures that do not return any result sets. Use the `ExecuteScalar` method to call stored procedures that return only a single value.

Use `HanaParameter` objects to pass parameters to a stored procedure.

Microsoft C# Stored Procedure Call with Parameters Example

This example shows two ways to call a stored procedure and pass it a parameter. The example uses a `HanaDataReader` to fetch the result set returned by the stored procedure.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
bool method1 = true;
HanaCommand cmd = new HanaCommand("", conn);
if (method1)
{
    cmd.CommandText = "DEMO.ShowProductInfo";
    cmd.CommandType = CommandType.StoredProcedure;
}
else
{
    cmd.CommandText = "call DEMO.ShowProductInfo(?)";
    cmd.CommandType = CommandType.Text;
}
HanaParameter param = cmd.CreateParameter();
param.HanaDbType = HanaDbType.Integer;
param.Direction = ParameterDirection.Input;
param.Value = 301;
cmd.Parameters.Add(param);
HanaDataReader reader = cmd.ExecuteReader();
reader.Read();
int ID = reader.GetInt32(0);
string name = reader.GetString(1);
string description = reader.GetString(2);
decimal price = reader.GetDecimal(6);
```

```
reader.Close();
listBox1.BeginUpdate();
listBox1.Items.Add("Name=" + name +
    " Description=" + description + " Price=" + price);
listBox1.EndUpdate();
conn.Close();
```

4.1.2 The .NET Core Provider

Use .NET Core to connect to SAP HANA.

You must choose the *.NET Core* option in the SAP HANA client install to use the SAP HANA driver for .NET Core. The driver is `Sap.Data.Hana.Core.v2.1.dll` and is available on Microsoft Windows, Linux, and macOS.

ADO.NET Support

The following ADO.NET functionality is not available for .NET Core in SAP HANA:

- Connecting via Microsoft ODBC data source entries
- Distributed transaction enlistment with a transaction coordinator
- Locating the unmanaged DLL via the ODBC HDBODBC registry entry
- HanaCredential class
- HanaConnection(string connectionString, HanaCredential credential) method
- HanaPermission class
- HanaFactory.CreatePermission method

Related Information

SAP Note 3136015

4.1.2.1 Run the .NET Core Examples

The SAP HANA client installation includes two example projects for .NET Core.

Prerequisites

The .NET Core SDK 2.1 or higher is installed.

Context

Example 1 Opens a database connection, executes a query (entered by the user), and then prints the result (column names and rows). It uses the HanaConnection, HanaCommand, and HanaDataReader classes.

Example 2 Opens a database connection, creates a table, inserts some rows (in batch mode), and then prints the result. It uses the HanaConnection, HanaCommand, and HanaDataAdapter classes.

Procedure

1. Open a command prompt (Microsoft Windows), bash shell (Linux), or Terminal (macOS).
2. For Linux/macOS, source the environment script: `source /usr/sap/hdbclient/hdbclienv.sh` or set the HDBDOTNETCORE environment variable to the location of the shared library: `export HDBDOTNETCORE=/usr/sap/hdbclient/dotnetcore`
3. Change the directory to the examples directory:

Operating System	Command
Microsoft Windows	`cd "C:\Program Files\SAP\hdbclient\examples\dotnetcore\Example1"`
Linux/macOS	`cd /usr/sap/hdbclient/examples/dotnetcore/Example1`

4. Edit the `Example1.csproj` file to verify which version of the .NET Core SDK being used. The following example uses version 2.1: `<target-framework>netcoreapp2.1<target-framework>`. This can be changed to target higher .NET Core SDK versions, for example version 3.1: `<target-framework>netcoreapp3.1<target-framework>`
5. Run the application via the .NET Core runtime: `dotnet run --project Example1.csproj`

4.1.3 The Microsoft .NET Framework Provider

Use the Microsoft .NET Framework to connect to SAP HANA.

4.1.3.1 Microsoft Entity Framework Support (SAP HANA Data Provider for Microsoft ADO.NET)

The data provider supports Microsoft Entity Framework 6.0.

To use Microsoft Entity Framework 6.0, you must add it to Microsoft Visual Studio by using Microsoft's NuGet Package Manager.

One of the new features of Microsoft Entity Framework is Code First. It enables a different development workflow: defining data model objects by simply writing Microsoft Visual Basic or Microsoft C# .NET classes mapping to database objects without ever having to open a designer or define an XML mapping file. Optionally, additional configuration can be performed by using data annotations or the Fluent API. Models can be used to generate a database schema or map to an existing database.

The following example creates new database objects by using the model:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Sap.Data.Hana;
namespace CodeFirstExample
{
    [Table( "EdmCategories", Schema = "DEMO" )]
    public class Category
    {
        public string CategoryID { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
        public virtual ICollection<Product> Products { get; set; }
    }
    [Table( "EdmProducts", Schema = "DEMO" )]
    public class Product
    {
        public int ProductId { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
        public string CategoryID { get; set; }
        public virtual Category Category { get; set; }
    }
    [Table( "EdmSuppliers", Schema = "DEMO" )]
    public class Supplier
    {
        [Key]
        public string SupplierCode { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
    }
    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }
        protected override void OnModelCreating( DbModelBuilder modelBuilder )
        {
            modelBuilder.Entity<Supplier>().Property( s => s.Name ).IsRequired();
        }
    }

    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new HanaConnectionFactory();
            Database.SetInitializer<Context>(
                new DropCreateDatabaseAlways<Context>() );
            using ( var db = new Context(
```

```
                "server=hana-db:30015;UserID=JSmith;password=secret" ) )
            {
                var query = db.Products.ToList();
            }
        }
    }
}
```

To build and run this example, the following assembly references must be added:

```
EntityFramework
Sap.Data.Hana.v4.5
System.ComponentModel.DataAnnotations
System.Data.Entity
```

Here is another example that maps to an existing database:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Sap.Data.Hana;
namespace CodeFirstExample
{
    [Table( "Customers", Schema = "DEMO" )]
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }
        public virtual ICollection<Contact> Contacts { get; set; }
    }
    [Table( "Contacts", Schema = "DEMO" )]
    public class Contact
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }
        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }
    public class Context : DbContext
    {
```

```
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }
        public DbSet<Contact> Contacts { get; set; }
        public DbSet<Customer> Customers { get; set; }
    }
    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new HanaConnectionFactory();
            Database.SetInitializer<Context>( null );
            using ( var db = new Context(
                "server=hana-db:30015;UserID=JSmith;password=secret" ) )
            {
                foreach ( var customer in db.Customers.ToList() )
                {
                    Console.WriteLine( "Customer - " + string.Format(
                      "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}",
                        customer.ID, customer.SurName, customer.GivenName,
                        customer.Street, customer.City, customer.State,
                        customer.Country, customer.PostalCode,
                        customer.Phone, customer.CompanyName ) );
                    foreach ( var contact in customer.Contacts )
                    {
                        Console.WriteLine( "    Contact - " + string.Format(
                          "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9},
{10}",
                            contact.ID, contact.SurName, contact.GivenName,
                            contact.Title,
                            contact.Street, contact.City, contact.State,
                            contact.Country, contact.PostalCode,
                            contact.Phone, contact.Fax ) );
                    }
                }
            }
        }
    }
}
```

There are some implementation detail differences between the Microsoft .NET Framework Data Provider for Microsoft SQL Server (SqlClient) and this data provider.

1. A new class HanaConnectionFactory (implements IDbConnectionFactory) is included. Set the Database.DefaultConnectionFactory to an instance of HanaConnectionFactory before creating any data model as shown below:

   ```
   Database.DefaultConnectionFactory = new HanaConnectionFactory();
   ```

2. The major principle of Microsoft Entity Framework Code First is coding by conventions. The Microsoft Entity Framework also does lots of things implicitly. Some code conventions do not make sense for database management systems like SAP HANA. There are some differences between Microsoft SQL Server and these database servers.
   - Microsoft SQL Server permits access to multiple databases with a single sign-on. SAP HANA permits a connection to one database at a time.
   - If the user creates a user-defined DbContext by using the parameterless constructor, then SqlClient connects to Microsoft SQL Server Express on the local computer using integrated security. The data provider connects to the default server using integrated login if the user has already created a login mapping.
   - SqlClient drops the existing database and creates a new database when the Microsoft Entity Framework calls DbDeleteDatabase or DbCreateDatabase (Microsoft SQL Server Express Edition

only). The data provider never drops or creates the database. It creates or drops the database objects (tables, relations, and constraints for example).

- The IDbConnectionFactory.CreateConnection method treats the string parameter nameOrConnectionString as a database name (initial catalog for Microsoft SQL Server) or a connection string. If the user does not provide the connection string for DbContext, then SqlClient automatically connects to the Microsoft SQL Express server on the local computer using the namespace of user-defined DbContext class as the initial catalog. For SAP HANA, that parameter can only contain a connection string. A database name is ignored and integrated login is used instead.

3. The Microsoft SQL Server SqlClient API maps a column with data annotation attribute TimeStamp to the Microsoft SQL Server data type timestamp/rowversion. The Microsoft SQL Server timestamp/rowversion data type is different from SAP HANA and most other RDBMS:
   - The Microsoft SQL Server timestamp/rowversion is binary(8). It is does not support a combined date and time value. SAP HANA supports a data type called timestamp that is equivalent to the Microsoft SQL Server datetime data type.
   - Microsoft SQL Server timestamp/rowversion values are guaranteed to be unique. SAP HANA timestamp values are not unique.
   - A Microsoft SQL Server timestamp/rowversion value changes every time the row is updated.

   The TimeStamp data annotation attribute is not supported by the data provider.
4. By default, Microsoft Entity Framework always sets the schema or owner name to dbo which is the default schema of Microsoft SQL Server. However, dbo is not appropriate for SAP HANA databases. For SAP HANA, specify the schema or owner name (DEMO for example) with the table name either by using data annotations or the Fluent API. Here is an example:

```
namespace CodeFirstTest
{
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }
        public virtual ICollection<Contact> Contacts { get; set; }
    }
    public class Contact
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }
        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }
    [Table( "Departments", Schema = "DEMO" )]
```

```
    public class Department
    {
        [Key()]
        public int DepartmentID { get; set; }
        public string DepartmentName { get; set; }
        public int DepartmentHeadID { get; set; }
    }
    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }
        public DbSet<Contact> Contacts { get; set; }
        public DbSet<Customer> Customers { get; set; }
        public DbSet<Department> Departments { get; set; }
        protected override void OnModelCreating( DbModelBuilder modelBuilder )
        {
            modelBuilder.Entity<Contact>().ToTable( "Contacts", "DEMO" );
            modelBuilder.Entity<Customer>().ToTable( "Customers", "DEMO" );
        }
    }
}
```

4.1.3.1.1 Use the Microsoft Code First to a New Database with the SAP HANA Data Provider Tutorial

You can use the Microsoft Code First to a New Database tutorial (based on the Microsoft .NET data provider) with the SAP HANA data provider. There are differences between the two providers that you must account for when following the tutorial.

Prerequisites

You must have Microsoft Visual Studio and the Microsoft .NET Framework installed on your computer.

Context

The Microsoft Code First to a New Database tutorial is designed for use with the Microsoft .NET data provider. The steps below provide guidance for using the SAP HANA data provider instead. Review these steps before attempting the tutorial.

Procedure

1. For Microsoft Entity Framework 6, make sure that the data provider that supports this version is installed. Make sure there are no running instances of Microsoft Visual Studio. At a command prompt with administrator privileges, do the following:

```
cd "c:\Program Files\sap\hdbclient\ado.net\v4.5"
```

```
SetupNet.v4.5.exe /i /hanalocation "c:\Program Files\sap\hdbclient\ado.net"
```

2. Before creating your Microsoft Visual Studio project, connect to the SAP HANA server and use the following SQL script to remove the Blogs, Posts, and Users tables, if you have already created them in another tutorial.

```
DROP INDEX "DEMO"."FK_BlogId";
DELETE FROM "DEMO"."Posts";
DELETE FROM "DEMO"."Blogs";
DROP TABLE "DEMO"."Posts";
DROP TABLE "DEMO"."Blogs";
DROP TABLE "DEMO"."Users";
```

3. Start Microsoft Visual Studio. The steps that follow were performed successfully using Microsoft Visual Studio 2013 and Microsoft Entity Framework 6.1.3.
4. One of the steps requires that you install the latest version of Microsoft Entity Framework 6 into the project, using the Nuget Package Manager.
   This step creates an `App.config` file that is not suitable for use with the data provider.
5. Replace the contents of `App.config` with the following.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://
go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <entityFramework>
    <defaultConnectionFactory type="Sap.Data.Hana.HanaConnectionFactory,
Sap.Data.Hana.v4.5, Version=1.0.100.0, Culture=neutral,
PublicKeyToken=0326b8ea63db4bc4">
        <parameters>
            <parameter value="server=hana-
server:30015;UserID=JSmith;password=secret" />
        </parameters>
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="Sap.Data.Hana"
type="Sap.Data.Hana.HanaProviderServices, Sap.Data.Hana.v4.5,
Version=1.0.100.0, Culture=neutral, PublicKeyToken=0326b8ea63db4bc4" />
    </providers>
  </entityFramework>
</configuration>
```

   Change the sample parameter value to a connection string that is suitable for connecting to your SAP HANA server.
6. Update all occurrences of the data provider version number in `App.config`. The version number should match the version of the data provider that you have currently installed.
7. Once you have updated `App.config`, you must build your project.
8. At the Create the Model step, include *Table* attributes so that DEMO is used for the schema name instead of `dbo` when the tables are created. Make sure that the DEMO schema exists in your database. Also set the *StringLength* attributes for `Name`, `Url` and `Title`. This setting prevents the use of the LOB data type

for the corresponding columns. Columns of type LOB cannot be used in any ORDER BY or WHERE clauses that may be generated.

```
    [Table("Blogs", Schema="DEMO")]
    public class Blog
    {
        public int BlogId { get; set; }
        [StringLength(200)]
        public string Name { get; set; }
        [StringLength(200)]
        public string Url { get; set; }
        public virtual List<Post> Posts { get; set; }
    }
    [Table("Posts", Schema = "DEMO")]
    public class Post
    {
        public int PostId { get; set; }
        [StringLength(200)]
        public string Title { get; set; }
        public string Content { get; set; }
        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }
```

Include the following *using* directives at the top of `Program.cs` so that the table attributes are recognized.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

9. Continue with the instructions in the remaining steps of the tutorial.

Results

You have built a Microsoft Entity Framework application that uses the Code First approach when the tables do not already exist in the database.

4.1.3.1.2 Use the Microsoft Code First to an Existing Database with the SAP HANA Data Provider Tutorial

You can use the Microsoft Code First to an Existing Database tutorial (based on the Microsoft .NET data provider) with the SAP HANA data provider. There are differences between the two providers that you must account for when following the tutorial.

Prerequisites

You must have Microsoft Visual Studio and the Microsoft .NET Framework installed on your computer.

Context

The Microsoft Code First to an Existing Database tutorial is designed for use with the Microsoft .NET data provider. The steps below provide guidance for using the SAP HANA data provider instead. Review these steps before attempting the tutorial.

Procedure

1. For Microsoft Entity Framework 6, make sure that the data provider that supports this version is installed. Make sure there are no running instances of Microsoft Visual Studio. At a command prompt with administrator privileges, do the following:

   ```
   cd "c:\Program Files\sap\hdbclient\ado.net\v4.5"
   SetupNet.v4.5.exe /i /hanalocation "c:\Program Files\sap\hdbclient\ado.net"
   ```

2. Before creating your Microsoft Visual Studio project, connect to the SAP HANA server and use the following SQL script to set up the Blogs, Posts, and Users tables. It is similar to the one presented in Microsoft tutorials but uses SAP HANA SQL statements and the DEMO schema owner instead of *dbo*.

   ```
   CREATE SCHEMA "DEMO";
   CREATE COLUMN TABLE "DEMO"."Blogs" (
       "BlogId" INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
       "Name" NVARCHAR (200) NULL,
       "Url"  NVARCHAR (200) NULL
   );
   CREATE COLUMN TABLE "DEMO"."Posts" (
       "PostId" INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
       "Title" NVARCHAR (200) NULL,
       "Content" NTEXT NULL,
       "BlogId" INTEGER NOT NULL
   );
   ALTER TABLE "DEMO"."Posts"
       ADD CONSTRAINT "FK_BlogId"
       FOREIGN KEY ("BlogId") REFERENCES "DEMO"."Blogs"("BlogId");
   INSERT INTO "DEMO"."Blogs" ("Name","Url")
   VALUES ('SAP HANA and In-Memory Computing', 'http://scn.sap.com/community/
   hana-in-memory');
   INSERT INTO "DEMO"."Blogs" ("Name","Url")
   VALUES ('SAP Business Trends', 'http://scn.sap.com/community/business-
   trends');
   COMMIT;
   CREATE TABLE "DEMO"."Users"
   (
       "Username" NVARCHAR(50) NOT NULL PRIMARY KEY,
       "DisplayName" NVARCHAR(50) NULL
   );
   ```

3. Start Microsoft Visual Studio. The steps that follow were performed successfully using Microsoft Visual Studio 2013 and Microsoft Entity Framework 6.1.3.
4. In the Microsoft Visual Studio *Server Explorer*, use the *Microsoft .NET Framework Data Provider for SAP HANA* to create a data connection. Select a suitable *Data source*. For example, set *Host name* to your SAP HANA server and port (hana-server:30015). Fill in the *User ID* and *Password* fields. The *Test Connection* button is used to ensure that a connection can be made to the database. Click *OK* when you are done.
5. Immediately after creating your new project, install the latest version of Microsoft Entity Framework 6 into the project, using the Nuget Package Manager.

This step creates an `App.config` file that is not suitable for use with the data provider.

6. Replace the contents of `App.config` with the following.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://
go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <entityFramework>
    <defaultConnectionFactory type="Sap.Data.Hana.HanaConnectionFactory,
Sap.Data.Hana.v4.5, Version=1.0.100.0, Culture=neutral,
PublicKeyToken=0326b8ea63db4bc4">
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="Sap.Data.Hana"
type="Sap.Data.Hana.HanaProviderServices, Sap.Data.Hana.v4.5,
Version=1.0.100.0, Culture=neutral, PublicKeyToken=0326b8ea63db4bc4" />
    </providers>
  </entityFramework>
</configuration>
```

7. Update all occurrences of the data provider version number in `App.config`. The version number should match the version of the data provider that you have currently installed.
8. Once you have updated `App.config`, you must build your project. This must be done before the Reverse Engineer Model step.
9. In the *Entity Data Model Wizard*, select the option to include sensitive data in the connection string.
10. Rather than select all tables, expand DEMO in *Tables* and select the Blogs and Posts tables.
11. Continue with the instructions in the remaining steps of the tutorial.

Results

You have built a Microsoft Entity Framework application that uses the Code First approach for tables that already exist in the database.

4.1.3.1.3 Use the Microsoft Database First to an Existing Database with the SAP HANA Data Provider Tutorial

You can use the Microsoft Existing Database tutorial (based on the Microsoft .NET data provider) with the SAP HANA data provider. There are differences between the two providers that you must account for when following the tutorial.

Prerequisites

You must have Microsoft Visual Studio and the Microsoft .NET Framework installed on your computer.

Context

The Microsoft Database First to an Existing Database tutorial is designed for use with the Microsoft .NET data provider. The steps below provide guidance for using the SAP HANA data provider instead. Review these steps before attempting the tutorial.

Procedure

1. For Microsoft Entity Framework 6, make sure that the data provider that supports this version is installed. Make sure there are no running instances of Microsoft Visual Studio. At a command prompt with administrator privileges, do the following:

   ```
   cd "c:\Program Files\sap\hdbclient\ado.net\v4.5"
   SetupNet.v4.5.exe /i /hanalocation "c:\Program Files\sap\hdbclient\ado.net"
   ```

2. Before creating your Microsoft Visual Studio project, connect to the SAP HANA server and use the following SQL script to set up the Blogs, Posts, and Users tables. It is similar to the one presented in the Microsoft tutorials but uses SAP HANA SQL statements and the DEMO schema owner instead of *dbo*.

   ```
   CREATE SCHEMA "DEMO";
   CREATE COLUMN TABLE "DEMO"."Blogs" (
       "BlogId" INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
       "Name" NVARCHAR (200) NULL,
       "Url"  NVARCHAR (200) NULL
   );
   CREATE COLUMN TABLE "DEMO"."Posts" (
       "PostId" INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
       "Title" NVARCHAR (200) NULL,
       "Content" NTEXT NULL,
       "BlogId" INTEGER NOT NULL
   );
   ALTER TABLE "DEMO"."Posts"
       ADD CONSTRAINT "FK_BlogId"
       FOREIGN KEY ("BlogId") REFERENCES "DEMO"."Blogs"("BlogId");
   ```

```
INSERT INTO "DEMO"."Blogs" ("Name","Url")
VALUES ('SAP HANA and In-Memory Computing', 'http://scn.sap.com/community/
hana-in-memory');
INSERT INTO "DEMO"."Blogs" ("Name","Url")
VALUES ('SAP Business Trends', 'http://scn.sap.com/community/business-
trends');
COMMIT;
CREATE TABLE "DEMO"."Users"
(
    "Username" NVARCHAR(50) NOT NULL PRIMARY KEY,
    "DisplayName" NVARCHAR(50) NULL
);
```

3. Start Microsoft Visual Studio. The steps that follow were performed successfully using Microsoft Visual Studio 2013 and Microsoft Entity Framework 6.1.3.
4. In the Microsoft Visual Studio *Server Explorer*, use the *Microsoft .NET Framework Data Provider for SAP HANA* to create a data connection. Select a suitable *Data source*. For example, set *Host name* to your SAP HANA server and port (hana-server:30015). Fill in the *User ID* and *Password* fields. The *Test Connection* button is used to ensure that a connection can be made to the database. Click *OK* when you are done.
5. Immediately after creating your new project, install the latest version of Microsoft Entity Framework 6 into the project, using the Nuget Package Manager.
   This step creates an `App.config` file that is not suitable for use with the data provider.
6. Replace the contents of `App.config` with the following.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://
go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <entityFramework>
    <defaultConnectionFactory type="Sap.Data.Hana.HanaConnectionFactory,
Sap.Data.Hana.v4.5, Version=1.0.100.0, Culture=neutral,
PublicKeyToken=0326b8ea63db4bc4">
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="Sap.Data.Hana"
type="Sap.Data.Hana.HanaProviderServices, Sap.Data.Hana.v4.5,
Version=1.0.100.0, Culture=neutral, PublicKeyToken=0326b8ea63db4bc4" />
    </providers>
  </entityFramework>
</configuration>
```

7. Update all occurrences of the data provider version number in `App.config`. The version number should match the version of the data provider that you have currently installed.
8. Once you have updated `App.config`, you must build your project. This must be done before the Reverse Engineer Model step.
9. In the *Microsoft Entity Data Model Wizard*, select the option to include sensitive data in the connection string.
10. Rather than select all tables, expand DEMO in *Tables* and select the Blogs and Posts tables.
11. Continue with the instructions in the remaining steps of the tutorial.
    If you used the *Update Wizard* to add the Users table, rebuild the project to see the changes to the model.

Results

You have built an Microsoft Entity Framework application that uses the Database First approach for tables that already exist in the database.

4.1.3.1.4 Use the Microsoft Model First to a New Database with the SAP HANA Data Provider Tutorial

You can use the Microsoft Model First tutorial (based on the Microsoft .NET data provider) with the SAP HANA data provider. There are differences between the two providers that you must account for when following the tutorial.

Prerequisites

You must have Microsoft Visual Studio and the Microsoft .NET Framework installed on your computer.

Context

The Microsoft Model First tutorial is designed for use with the Microsoft .NET data provider. The steps below provide guidance for using the SAP HANA data provider instead. Review these steps before attempting the tutorial.

Procedure

1. For Microsoft Entity Framework 6, make sure that the data provider that supports this version is installed. Make sure there are no running instances of Microsoft Visual Studio. At a command prompt with administrator privileges, do the following:

   ```
   cd "c:\Program Files\sap\hdbclient\ado.net\v4.5"
   SetupNet.v4.5.exe /i /hanalocation "c:\Program Files\sap\hdbclient\ado.net"
   ```

2. Before creating your Microsoft Visual Studio project, connect to the SAP HANA server and use the following SQL script to remove the Blogs, Posts, and Users tables if you created them in another tutorial.

   ```
   DROP INDEX "DEMO"."FK_BlogId";
   DELETE FROM "DEMO"."Posts";
   DELETE FROM "DEMO"."Blogs";
   DROP TABLE "DEMO"."Posts";
   DROP TABLE "DEMO"."Blogs";
   DROP TABLE "DEMO"."Users";
   ```

3. Start Microsoft Visual Studio.

4. In the Microsoft Visual Studio *Server Explorer*, use the *Microsoft .NET Framework Data Provider for SAP HANA* to create a data connection. Select a suitable *Data source*. For example, set *Host name* to your SAP HANA server and port (hana-server:30015). Fill in the *User ID* and *Password* fields. The *Test Connection* button is used to ensure that a connection can be made to the database. Click *OK* when you are done.
5. Immediately after creating your new project, install the latest version of Microsoft Entity Framework 6 into the project, using the Nuget Package Manager.
   This step creates an `App.config` file that is not suitable for use with the data provider.
6. Replace the contents of `App.config` with the following.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://
go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <entityFramework>
    <defaultConnectionFactory type="Sap.Data.Hana.HanaConnectionFactory,
Sap.Data.Hana.v4.5, Version=1.0.100.0, Culture=neutral,
PublicKeyToken=0326b8ea63db4bc4">
        <parameters>
            <parameter
value="server=ykfl00540320a.amer.global.corp.sap:30015;UserID=SYSTEM;password=
manager" />
        </parameters>
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="Sap.Data.Hana"
type="Sap.Data.Hana.HanaProviderServices, Sap.Data.Hana.v4.5,
Version=1.0.100.0, Culture=neutral, PublicKeyToken=0326b8ea63db4bc4" />
    </providers>
  </entityFramework>
</configuration>
```

7. Update all occurrences of the data provider version number in `App.config`. The version number should match the version of the data provider that you have currently installed.
8. Once you have updated `App.config`, you must build your project. This must be done before using the Microsoft Entity Framework Designer.
9. Before the Generating the Database step, open the *Properties* of `BloggingModel.edmx [Diagram1]` (this is your design form) and set *Database Schema Name* to DEMO and set *DDL Generation Template* to `SSDLToHana.tt (VS)`.
10. In the *Generate Database Wizard*, select the option to include sensitive data in the connection string.
11. Continue with the instructions in the remaining steps of the tutorial.

Results

You have built a Microsoft Entity Framework application that uses the Model First approach when the tables do not already exist in the database.

4.1.3.2 Entity Framework Core Support (SAP HANA Data Provider for Microsoft ADO.NET)

SAP HANA supports Entity Framework Core so that you can develop applications.

The supported Entity Framework versions are Entity Framework Core 6 and 7.

SAP HANA supports version 2.x and higher.

Installation

The SAP HANA Entity Framework Core Provider is shipped with the SAP HANA database client. It contains one .NET assembly (`Sap.EntityFrameworkCore.Hana.<version>.0.dll`) which is located in `<hdbclient>\dotnetcore\<version>`.

The provider references `Sap.Data.Hana.Core.v2.1.dll`. If your project is a direct dependency of the SAP HANA .NET Core Provider, it must reference `Sap.Data.Hana.Core.v2.1.dll`.

Entity Framework Core 7

The Entity Framework Core 7 driver supports columns that contain JSON documents. This allows mapping of aggregates built from .NET types to JSON documents. In Entity Framework Core, JSON columns can be used to store and query JSON data in an SAP HANA database. Queries into JSON columns work the same as querying any other aggregate type in Entity Framework Core.

The commands `SaveChanges` and `SaveChangesAsync` are used to make updates to a JSON column. For extensive changes, the entire document is updated. However, updating a sub-document is not supported.

Entity Framework Core 7 for NetTopologySuite

Spatial data represents the physical location and the shape of objects. SAP HANA databases provide support for this type of data so it can be queried alongside other data. Common scenarios include: querying for objects within a given distance from a location or selecting the object whose border contains a given location. EF Core supports mapping to spatial data types using the NetTopologySuite spatial library.

The SAP HANA EF Core 7 provider for NetTopologySuite contains the .NET Core-compatible assemblies with Entity Framework Core-related functionality for SAP HANA. It allows you to interact with SAP HANA databases via the widely-used .NET O/RM from Microsoft, and use familiar LINQ syntax to express queries for spatial data.

Related Information

Entity Framework Core
Basic Save

4.1.3.2.1 Example: Use the SAP HANA Entity Framework Core Database Provider

Use the SAP HANA Entity Framework Core database provider to create and query blogs.

Prerequisites

- You must have the Microsoft .NET 6 or later installed on your computer.
- You can use either the .NET SDK (6 or later) or the Microsoft Visual Studio (2019 or 2022) to build the example.

Context

This example demonstrates how to use modelling, using the Blog and Post entity classes, with a context object to interact with the underlying SAP HANA database to create two new blogs, each with a new post, and then look for blogs and posts matching specific criteria.

Procedure

1. Configure the project.

   Create a C# Console Application that targets the .NET 6 Framework. Add the following packages to the `.csproj` file:

   - Microsoft.EntityFrameworkCore 6
   - Microsoft.EntityFrameworkCore.Relational 6

   Ensure your `.csproj` file contains `<"Sap.EntityFrameworkCore.Hana.v6.0" />`.

   To build and run this example, import the following namespaces:

   ```
   using Microsoft.EntityFrameworkCore;
   using Sap.EntityFrameworkCore.Hana;
   ```

2. Create the Entity Framework Core model.

   ```
   public class HanaDbContext : DbContext
   ```

```
{
    public HanaDbContext(bool createTables) : base()
    {
        if (createTables)
        {
            // Delete the existing database tables and re-create new tables.
            Database.EnsureDeleted();
            Database.EnsureCreated();
        }
    }
    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        optionsBuilder.UseHana("Server=my-
server:30015;UserName=SYSTEM;Password=password");
    }
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public int Rating { get; set; }
    public string? Description { get; set; }
    public List<Post> Posts { get; set; }
}
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

> **Note**
>
> Specify the UseHana method to configure the context to connect to an SAP HANA database with the connection string.

3. Create the query entities.

   Add the following code to the Main function:

```
using (var context = new HanaDbContext(true))
{
    context.AddRange(
        new Blog
        {
            Url = "https://blogs.sap.com/",
            Rating = 4,
            Description = "All Blog Posts",
            Posts = new List<Post>
            {
                new Post {Title = "New client interface features", Content =
"Posts about new features in the SAP HANA client interfaces." }
            }
        },
        new Blog
        {
            Url = "https://community.sap.com/",
            Rating = 5,
            Description = "SAP Community",
            Posts = new List<Post>
            {
```

```
                new Post {Title = "SAP HANA client installation", Content =
"Client interfaces for connecting applications as part of the SAP HANA client
software package." }
            }
        });
    var count = context.SaveChanges();
    Console.WriteLine($"Saved {count} entities to the database.");
    Console.WriteLine("List blogs with Rating above 3.");
    var blogs = context.Blogs.Where(b => b.Rating > 3).ToList();
    foreach (var blog in blogs)
        Console.WriteLine($"Url = {blog.Url}, Rating = {blog.Rating},
Description = {blog.Description}");
    Console.WriteLine("List posts contain interfaces.");
    var posts = context.Posts.Where(p =>
p.Content.Contains("interfaces")).ToList();
    foreach (var post in posts)
        Console.WriteLine($"Title = {post.Title}, Content = {post.Content}");
}
```

4. Build and run the example.

   Use dotnet SDK or Visual Studio to build and run the example.

Related Information

Entity Framework Core

4.1.3.2.2 Example: Use the SAP HANA Entity Framework Core Database Provider for NetTopologySuite

Use the SAP HANA Entity Framework Core database provider for NetTopologySuite to create and query spatial data.

Prerequisites

- You must have the Microsoft .NET 6 or later installed on your computer.
- You can use either the .NET SDK (6 or later) or the Microsoft Visual Studio (2019 or 2022) to build the example.

Context

This example demonstrates how to use modelling, the City entity class, with a context object to interact with the underlying SAP HANA database to create new cities, each with a spatial location, and then look for cities matching specific criteria.

> **Note**
>
> Some NetTopologySuite methods, for example CurveToLine, are not supported by the SAP HANA database.

Procedure

1. Configure the project:

   Create a C# console application that targets the .NET 6 Framework. Add the following packages to the `.csproj` file:

   - Microsoft.EntityFrameworkCore 7
   - Microsoft.EntityFrameworkCore.Relational 7
   - NetTopologySuite 2.5

   > **Note**
   >
   > Ensure your `.csproj` file contains `<"Sap.EntityFrameworkCore.Hana.v7.0" />` and `"<Sap.EntityFrameworkCore.Hana.NetTopologySuite.v7.0" />`.

   Build and run this example by importing the following namespaces:

   ```
   using Microsoft.EntityFrameworkCore;
   using NetTopologySuite.Geometries;
   using System.ComponentModel.DataAnnotations.Schema;
   using Sap.EntityFrameworkCore;
   using Sap.EntityFrameworkCore.Hana;
   ```

2. 2. Create the Entity Framework Core model.

   ```
   public class HanaDbContext : DbContext
   {
       public HanaDbContext(DbContextOptions opts, bool create) : base(opts)
       {
           if (create)
           {
               this.Database.EnsureDeleted();
               this.Database.EnsureCreated();
           }
       }
       public DbSet<City> Cities { get; set; }
   }
   public class City
   {
       public int Id { get; set; }
       public string CityName { get; set; }
       [Column(TypeName = "ST_GEOMETRY(0)")]
       public Geometry Location { get; set; }
   ```

   > **Note**
   >
   > Specify the UseHana method to configure the context to connect to an SAP HANA database with the connection string.

3. Create the database and add some cities.

   Add the following code to the Main function:

```
        var srid = 0;
        var factory =
NetTopologySuite.NtsGeometryServices.Instance.CreateGeometryFactory(srid:
srid);
        var ottawa = factory.CreatePoint(new
NetTopologySuite.Geometries.Coordinate(-75.695000, 45.424721));
        City vancouver = new City()
        {
            CityName = "Vancouver",
            Location = new Point(49.246292, -123.116226) { SRID = srid }
        };
        City toronto = new City()
        {
            CityName = "Toronto",
            Location = new Point(-79.347015, 43.651070) { SRID = srid }
        };
        City montreal = new City()
        {
            CityName = "Montreal",
            Location = new Point(-73.561668, 45.508888) { SRID = srid }
        };
        City quebec = new City()
        {
            CityName = "Quebec",
            Location = new Point(-71.254028, 46.829853) { SRID = srid }
        };
        DbContextOptionsBuilder opt = new DbContextOptionsBuilder();
        opt.UseHana(@"Server= my-server:30015
;UserName=SYSTEM;Password=manager",
            x => x.UseNetTopologySuite());
        HanaDbContext db1 = new HanaDbContext(opt.Options, true);
        db1.Cities.AddRange(vancouver, toronto, montreal, quebec);
        var rows = db1.SaveChanges();
        Console.WriteLine($"Added {rows} cities");
```

4. Query the City entities.

   Add the following code to the Main function:

```
    // List all cities, order by distance from Ottawa.
        HanaDbContext db2 = new HanaDbContext(opt.Options, false);
        var query = db2.Cities.OrderBy(c =>
c.Location.Distance(ottawa)).ToList();
        foreach (City c in query)
        {
            Console.WriteLine($"Id = {c.Id}, CityName = {c.CityName},
Location = {c.Location.ToText()}");
        }
```

5. Build and run the example using dotnet SDK or Visual Studio.

4.1.3.2.3 Example: Use the SAP HANA Entity Framework Core Database Provider for Scaffolding (Reverse Engineering)

Scaffolding (reverse engineering) is the process of generating code of entity type classes and a DbContext class based on a database schema. It can be performed using the dotnet ef dbcontext scaffold command of the .NET Command-line Interface (CLI) tools.

Prerequisites

- You must have the Microsoft .NET 6 or later installed on your computer.
- Before scaffolding, you'll need to install the .NET CLI tools. For instructions, see https://learn.microsoft.com/en-us/ef/core/cli/dotnet .

Context

This example demonstrates how to use the SAP HANA Entity Framework Core database provider for scaffolding an Entity Framework Core model.

Procedure

1. Connect to the SAP HANA server. Use the following SQL script to set up the Blogs, Posts, and Users tables:

```
CREATE SCHEMA "DEMO";
CREATE COLUMN TABLE "DEMO"."Blogs" (
    "BlogId" INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    "Name" NVARCHAR (200) NULL,
    "Url"  NVARCHAR (200) NULL
);
CREATE COLUMN TABLE "DEMO"."Posts" (
    "PostId" INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    "Title" NVARCHAR (200) NULL,
    "Content" NTEXT NULL,
    "BlogId" INTEGER NOT NULL
);
ALTER TABLE "DEMO"."Posts"
    ADD CONSTRAINT "FK_BlogId"
    FOREIGN KEY ("BlogId") REFERENCES "DEMO"."Blogs"("BlogId");
CREATE TABLE "DEMO"."Users"
(
    "Username" NVARCHAR(50) NOT NULL PRIMARY KEY,
    "DisplayName" NVARCHAR(50) NULL
);
```

2. Create .NET C# console app targets on .NET 6.0 or later.

   Add the following assemblies and packages to the project dependencies:

- Sap.Data.Hana.Core.v2.1.dll
- Sap.EntityFrameworkCore.Hana.v7.0.dll
- Microsoft.EntityFrameworkCore 7
- Microsoft.EntityFrameworkCore.Design 7
- Microsoft.EntityFrameworkCore.Relational 7

3. Build the project.
4. Open a command prompt and `CD <project directory>`. Then, run the .NET CLI command to scaffold the Entity Framework Core model.

   Here are some examples:

   ```
   dotnet ef dbcontext scaffold "Server=<my-
   server:30015>;uid=<uid>;pwd=<pwd>;Current Schema=DEMO"
   Sap.EntityFrameworkCore.Hana.v7.0
   ```

   ```
   dotnet ef dbcontext scaffold "Server=<my-server:30015>;uid=<uid>;pwd=<pwd>"
   Sap.EntityFrameworkCore.Hana.v7.0 --schema DEMO
   ```

   ```
   dotnet ef dbcontext scaffold "Server=<my-server:30015>;uid=<uid>;pwd=<pwd>"
   Sap.EntityFrameworkCore.Hana.v7.0 --table Blogs --table Posts –-table Users
   ```

5. Add “using Sap.EntityFrameworkCore.Hana;” to `C55Context.cs`. Build the project.

Related Information

Entity Framework Core

4.1.3.3 The DLL for the SAP HANA Data Provider for Microsoft ADO.NET

When the data provider is loaded by a Microsoft .NET application (usually when making an SAP HANA database server connection by using `HanaConnection`), it loads an additional DLL containing the data provider's unmanaged code.

The unmanaged DLL is installed when you install the SAP HANA client software into the `ado.net` subdirectory. There are two DLL variants:

- `libadonetHDB.dll` (64-bit)
- `libadonetHDB32.dll` (32-bit)

To locate the current version of the DLL file, the `SetupNetv3.5.exe` or the `SetupNetv4.5.exe` ADO.NET client installer creates an environment variable `HDBADONET` with the path to the current version of the DLL at the time of installation.

The data provider locates the `libadonetHDB.dll` file that matches the version number of the data provider and the bitness of the Microsoft .NET application when it is loaded.

The data provider obtains the file path for the DLL using the following strategy:

- Use the current working directory of the application
- Use the current application executable location
- Use the HDBADONET environment variable path location
- Use the file path for the installed SAP HANA ODBC driver

For 64-bit Microsoft Windows platforms:	For 64-bit Microsoft .NET applications, the data provider obtains the file path from the `Driver` setting in the registry entry for the 64-bit SAP HANA ODBC driver (`HKLM\SOFTWARE\ODBC\ODBCINST.INI\HDBODBC`). For 32-bit Microsoft .NET applications and Microsoft Visual Studio, the data provider obtains the file path from the `Driver` setting in the registry entry for the 32-bit SAP HANA ODBC driver (`HKLM\SOFTWARE\Wow6432Node\ODBC\ODBCINST.INI\HDBODBC32`).
For 32-bit Microsoft Windows platforms:	For 32-bit Microsoft .NET applications and Microsoft Visual Studio, the data provider obtains the file path from the `Driver` setting in the registry entry for the 32-bit SAP HANA ODBC driver (`HKLM\SOFTWARE\ODBC\ODBCINST.INI\HDBODBC32`).

- Use the default SAP HANA client installation directory path `C:\Program Files\SAP\hdbclient` or `C:\Program Files (x86)\SAP\hdbclient`
- Search the `PATH` environment variable

For example, when connecting to the SAP HANA data source, if the unmanaged DLL cannot be located then a TypeInitializationException is returned (this includes an InnerException which is a FileNotFoundException). The message associated with this exception says, "Cannot find a matching `libadonetHDB.dll` with version X.Y.Z. - check the location in the HDBADONET or PATH environment variables."

4.1.3.4 Data Access and Manipulation (SAP HANA Data Provider for Microsoft ADO.NET)

With the data provider, there are two ways that you can access data: by using the `HanaCommand` class or the `HanaDataAdapter` class.

`HanaCommand` object	The `HanaCommand` object is the recommended way of accessing and manipulating data in Microsoft .NET. The `HanaCommand` object allows you to execute SQL statements that retrieve or modify data directly from the SAP HANA database server. Using the `HanaCommand` object, you can issue SQL statements and call stored procedures directly against the SAP HANA database server. Within a `HanaCommand` object, a `HanaDataReader` is used to return read-only result sets from a query or stored procedure. The `HanaDataReader` returns one row at a time. Using the `HanaCommand` object allows you to group your changes into transactions rather than operating in autocommit mode. When you use the `HanaTransaction` object, locks are placed on the rows so that other users cannot modify them.

`HanaDataAdapter` object	The `HanaDataAdapter` object retrieves the entire result set into a `DataSet`. A `DataSet` is a disconnected store for data that is retrieved from an SAP HANA database server. You can then edit the data in the `DataSet` and when you are finished, the `HanaDataAdapter` object updates the SAP HANA database server with the changes made to the `DataSet`. When you use the `HanaDataAdapter`, there is no way to prevent other users from modifying the rows in your `DataSet`. You must include logic within your application to resolve any conflicts that occur.

The performance of a `HanaDataReader` object is always significantly better than a `HanaDataAdapter` object.

4.1.3.4.1 BLOBs (SAP HANA Data Provider for Microsoft ADO.NET)

When fetching long string values or binary data, there are methods that you can use to fetch the data in pieces. For binary data, use the `GetBytes` method, and for string data, use the `GetChars` method.

Otherwise, BLOB data is treated in the same manner as any other data you fetch from the SAP HANA database server.

Microsoft C# GetChars BLOB Example

This example reads three columns from a result set. The first two columns are integers, while the third column is a CLOB. The length of the third column is computed by reading this column with the `GetChars` method in chunks of 100 characters.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
HanaCommand cmd = new HanaCommand(
    "SELECT * FROM DEMO.MarketingInformation", conn);
HanaDataReader reader = cmd.ExecuteReader();
int idValue;
int productIdValue;
int length = 100;
char[] buf = new char[length];
while (reader.Read())
{
    idValue = reader.GetInt32(0);
    productIdValue = reader.GetInt32(1);
    long blobLength = 0;
    long charsRead;
    while ((charsRead = reader.GetChars(2, blobLength, buf, 0, length))
            == (long)length)
    {
        blobLength += charsRead;
    }
    blobLength += charsRead;
}
reader.Close();
conn.Close();
```

4.1.3.4.2 HanaCommand: Fetch a Result Set Schema by Using GetSchemaTable

You can obtain schema information about columns in a result set by using the `GetSchemaTable` method.

The `GetSchemaTable` method of the `HanaDataReader` class obtains information about the current result set. The `GetSchemaTable` method returns the standard Microsoft .NET `DataTable` object, which provides information about all the columns in the result set, including column properties.

Microsoft C# Schema Information Example

This example obtains information about a result set by using the `GetSchemaTable` method and binds the `DataTable` object to the datagrid on the screen.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
HanaCommand cmd = new HanaCommand("SELECT * FROM DEMO.Employees", conn);
HanaDataReader reader = cmd.ExecuteReader();
DataTable schema = reader.GetSchemaTable();
reader.Close();
conn.Close();
dataGridView1.DataSource = schema;
```

4.1.3.4.3 HanaCommand: Fetch Data by Using ExecuteReader and ExecuteScalar

The `HanaCommand` object allows you to execute a SQL statement or call a stored procedure against an SAP HANA database server. Use the `ExecuteReader` or `ExecuteScalar` methods to retrieve data from the SAP HANA database server.

ExecuteReader	Issues a SQL query that returns a result set. This method uses a forward-only, read-only cursor. You can loop quickly through the rows of the result set in one direction.
ExecuteScalar	Issues a SQL query that returns a single value. This value can be the first column in the first row of the result set, or a SQL statement that returns an aggregate value such as COUNT or AVG. This method uses a forward-only, read-only cursor.

When using the `HanaCommand` object, you can use the `HanaDataReader` to retrieve a result set that is based on a join. However, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins.

When using the `HanaDataReader`, there are several `Get` methods available that you can use to return the results in the specified data type.

Microsoft C# ExecuteReader Example

The following Microsoft C# code opens a connection to an SAP HANA database server and uses the ExecuteReader method to create a result set containing the last names of employees in the Employees table:

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>");
conn.Open();
HanaCommand cmd = new HanaCommand("SELECT Surname FROM DEMO.Employees", conn);
HanaDataReader reader = cmd.ExecuteReader();
listEmployees.BeginUpdate();
while (reader.Read())
{
    listEmployees.Items.Add(reader.GetString(0));
}
listEmployees.EndUpdate();
reader.Close();
conn.Close();
```

Microsoft Visual Basic ExecuteReader Example

The following Microsoft Visual Basic code opens a connection to an SAP HANA database server and uses the `ExecuteReader` method to create a result set containing the last names of employees in the Employees table:

```
Dim conn As New HanaConnection( _
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>")
Dim cmd As New HanaCommand("SELECT Surname FROM DEMO.Employees", conn)
Dim reader As HanaDataReader
conn.Open()
reader = cmd.ExecuteReader()
ListEmployees.BeginUpdate()
Do While (reader.Read())
    ListEmployees.Items.Add(reader.GetString(0))
Loop
ListEmployees.EndUpdate()
reader.Close()
conn.Close()
```

Microsoft C# ExecuteScalar Example

The following Microsoft C# code opens a connection to an SAP HANA database server and uses the `ExecuteScalar` method to obtain a count of the number of male employees in the Employees table:

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>");
conn.Open();
HanaCommand cmd = new HanaCommand(
    "SELECT COUNT(*) FROM DEMO.Employees WHERE Sex = 'M'", conn );
Int64 count = (Int64)cmd.ExecuteScalar();
textBox1.Text = count.ToString();
conn.Close();
```

4.1.3.4.4 HanaCommand: Insert, Delete, and Update Rows by Using ExecuteNonQuery

To perform an insert, update, or delete with a `HanaCommand` object, use the `ExecuteNonQuery` method. The `ExecuteNonQuery` method issues a query (SQL statement or stored procedure) that does not return a result set.

You can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins.

To set the isolation level for a SQL statement, use the `HanaCommand` object as part of a `HanaTransaction` object. When you modify data without a `HanaTransaction` object, the data provider operates in autocommit mode and any changes that you make are applied immediately.

Microsoft C# ExecuteNonQuery DELETE and INSERT Example

This example opens a connection to an SAP HANA database server and uses the `ExecuteNonQuery` method to remove all departments whose ID is greater than or equal to 600 and then adds two new rows to the Departments table. It displays the updated table in a datagrid.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>");
conn.Open();
HanaCommand deleteCmd = new HanaCommand(
    "DELETE FROM DEMO.Departments WHERE DepartmentID >= 600",
    conn);
deleteCmd.ExecuteNonQuery();
HanaCommand insertCmd = new HanaCommand(
    "INSERT INTO DEMO.Departments(DepartmentID, DepartmentName) " +
    "VALUES( ?, ? )",
    conn );
HanaParameter parm = new HanaParameter();
parm.HanaDbType = HanaDbType.Integer;
insertCmd.Parameters.Add( parm );
parm = new HanaParameter();
parm.HanaDbType = HanaDbType.NVarChar;
insertCmd.Parameters.Add( parm );
insertCmd.Parameters[0].Value = 600;
insertCmd.Parameters[1].Value = "Eastern Sales";
int recordsAffected = insertCmd.ExecuteNonQuery();
insertCmd.Parameters[0].Value = 700;
insertCmd.Parameters[1].Value = "Western Sales";
recordsAffected = insertCmd.ExecuteNonQuery();
HanaCommand selectCmd = new HanaCommand(
    "SELECT * FROM DEMO.Departments", conn );
HanaDataReader dr = selectCmd.ExecuteReader();
dataGridView1.DataSource = dr;
dr.Close();
conn.Close();
```

Microsoft C# ExecuteNonQuery UPDATE Example

This example opens a connection to an SAP HANA database server and uses the `ExecuteNonQuery` method to update the DepartmentName column to Engineering in all rows of the Departments table where the DepartmentID is 100. It displays the updated table in a datagrid.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>");
conn.Open();
HanaCommand updateCmd = new HanaCommand(
    "UPDATE DEMO.Departments SET DepartmentName = 'Engineering' " +
    "WHERE DepartmentID = 100", conn );
int recordsAffected = updateCmd.ExecuteNonQuery();
HanaCommand selectCmd = new HanaCommand(
    "SELECT * FROM DEMO.Departments ORDER BY DepartmentID", conn );
HanaDataReader dr = selectCmd.ExecuteReader();
dataGridView1.DataSource = dr;
dr.Close();
conn.Close();
```

4.1.3.4.5 HanaDataAdapter Overview

The `HanaDataAdapter` retrieves a result set into a `DataTable`.

A `DataSet` is a collection of tables (DataTables) and the relationships and constraints between those tables. The `DataSet` is built into the Microsoft .NET Framework, and is independent of the data provider that is used to connect to your SAP HANA database server.

When you use the `HanaDataAdapter`, you must be connected to the SAP HANA database server to fill a `DataTable` and to update the table on the SAP HANA database server with changes made to the `DataTable`. However, once the `DataTable` is filled, you can modify the `DataTable` while disconnected from the SAP HANA database server.

If you do not want to apply your changes to the SAP HANA database server right away, then you can write the `DataSet`, including the data and/or the schema, to an XML file by using the `WriteXml` method. Then, you can apply the changes at a later time by loading a `DataSet` with the `ReadXml` method. The following code shows two examples.

```
ds.WriteXml("Employees.xml");
ds.WriteXml("EmployeesWithSchema.xml", XmlWriteMode.WriteSchema);
```

For more information, see the Microsoft .NET Framework documentation for `WriteXml` and `ReadXml`.

When you call the `Update` method to apply changes from the `DataSet` to the SAP HANA database server, the `HanaDataAdapter` analyzes the changes that have been made and then invokes the appropriate statements, INSERT, UPDATE, or DELETE, as necessary. When you use the DataSet, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins. If another user has a lock on the row you are trying to update, then the application throws an exception.

> ⚠ Caution
>
> You do not have to be connected to the SAP HANA database server when you make changes to the `DataSet`. Your application does not have any locks on these rows in the table. When changes from the

`DataSet` are applied to the SAP HANA database server, your application must resolve any conflicts that may occur if another user changes the data during the fetch-update interval.

Resolving Conflicts When Using the HanaDataAdapter

When you use the `HanaDataAdapter`, no locks are placed on the rows in the database table. This behavior means there is the potential for conflicts to arise when you apply changes from the `DataSet` to the database table. Include logic in your application to resolve or log conflicts that arise.

Some of the conflicts that your application logic can address include:

Unique primary keys	If two users insert new rows into a table, then each row must have a unique primary key.
Updates made to the same value	If two users modify the same value, then your application needs to include logic to determine which value is correct.
Schema changes	If a user modifies the schema of a table you have updated in the `DataSet`, then the update fails when you apply the changes to the SAP HANA database server.
Data concurrency	Concurrent applications need to see a consistent set of data. The `HanaDataAdapter` does not place a lock on rows that it fetches, so another user can update a value in the SAP HANA database server once you have retrieved the `DataSet`.

You can avoid many of these potential problems by using the `HanaCommand`, `HanaDataReader`, and `HanaTransaction` objects to apply changes to the SAP HANA database server. The `HanaTransaction` object is recommended because it allows you to set the isolation level for the transaction and it places locks on the rows so that other users cannot modify them.

To simplify the process of conflict resolution, you can design your INSERT, UPDATE, or DELETE statement to be a stored procedure call. By including INSERT, UPDATE, and DELETE statements in stored procedures, you can catch the error if the operation fails. In addition to the statement, you can add error handling logic to the stored procedure so that if the operation fails the appropriate action is taken, such as recording the error to a log file, or trying the operation again.

4.1.3.4.5.1 HanaDataAdapter: Fetch Data into a DataTable by Using Fill

The `HanaDataAdapter` allows you to view a result set by using the `Fill` method to fill a `DataTable` with the results from a query and then binding the `DataTable` to a display grid.

When setting up a `HanaDataAdapter`, you can specify a SQL statement that returns a result set. When `Fill` is called to populate a `DataTable`, all the rows are fetched in one operation by using a forward-only, read-only cursor. Once all the rows in the result set have been read, the cursor is closed. Changes made to the rows in a `DataTable` can be reflected to the SAP HANA database server by using the `Update` method.

You can use the `HanaDataAdapter` object to retrieve a result set that is based on a join. However, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins.

> ⚠ Caution
>
> Any changes you make to a `DataTable` are made independently of the original database table. Your application does not have any locks on these rows in the table. When changes from the `DataTable` are applied to the database table, your application must resolve any conflicts that occur if another user changed the data during the fetch-update interval.

Microsoft C# HanaDataAdapter Fill Example Using a DataTable

This example shows how to fill a `DataTable` by using the `HanaDataAdapter`. It creates a new `HanaDataAdapter` object and a new `DataTable` object named Results. The `HanaDataAdapter` `Fill` method is used to fill the `DataTable` with the results of the query. The `DataTable` is then bound to the grid on the screen.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
HanaDataAdapter da = new HanaDataAdapter(
    "SELECT * FROM DEMO.Employees ORDER BY EmployeeID", conn);
DataTable dt = new DataTable("Results");
da.Fill(dt);
conn.Close();
dataGridView1.DataSource = dt;
```

Microsoft C# HanaDataAdapter Fill Example Using a DataSet

This example shows how to fill a `DataTable` by using the `HanaDataAdapter`. It creates a new `DataSet` object and a new `HanaDataAdapter` object. The `HanaDataAdapter` `Fill` method is used to create a `DataTable` table named Results in the `DataSet` and then fill it with the results of the query. The Results `DataTable` is then bound to the grid on the screen.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
DataSet ds = new DataSet();
HanaDataAdapter da = new HanaDataAdapter(
    "SELECT * FROM DEMO.Employees ORDER BY EmployeeID", conn);
da.Fill(ds, "Results");
conn.Close();
dataGridView1.DataSource = ds.Tables["Results"];
```

4.1.3.4.5.2 HanaDataAdapter: Format a DataTable by Using FillSchema

The `HanaDataAdapter` allows you to configure the schema of a `DataTable` to match that of a specific query by using the `FillSchema` method. The attributes of the columns in the `DataTable` match those of the `SelectCommand` of the `HanaDataAdapter` object.

Unlike the `Fill` method, no rows are stored in the `DataTable`.

Microsoft C# HanaDataAdapter FillSchema Example Using a DataTable

This example shows how to use the `FillSchema` method to set up a new `DataTable` object with the same schema as a result set. The Additions `DataTable` is then bound to the grid on the screen.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
HanaDataAdapter da = new HanaDataAdapter(
    "SELECT * FROM DEMO.Employees ORDER BY EmployeeID", conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
conn.Close();
dataGridView1.DataSource = dt;
```

Microsoft C# HanaDataAdapter FillSchema Example Using a DataSet

This example shows how to use the `FillSchema` method to set up a new `DataTable` object with the same schema as a result set. The `DataTable` is added to the `DataSet` by using the `Merge` method. The Additions `DataTable` is then bound to the grid on the screen.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
HanaDataAdapter da = new HanaDataAdapter(
    "SELECT * FROM DEMO.Employees ORDER BY EmployeeID", conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
DataSet ds = new DataSet();
ds.Merge(dt);
conn.Close();
dataGridView1.DataSource = ds.Tables["Additions"];
```

4.1.3.4.5.3 HanaDataAdapter: Insert Rows by Using Add and Update

The `HanaDataAdapter` allows you to insert rows into a database table by using the `Add` and `Update` methods.

Microsoft C# HanaDataAdapter Insert Example

This example shows how to use the `Update` method of `HanaDataAdapter` to add rows to a table. The example fetches the Departments table into a `DataTable` by using the `SelectCommand` property and the `Fill` method of the `HanaDataAdapter`. It then adds two new rows to the `DataTable` and updates the Departments table from the `DataTable` by using the `InsertCommand` property and the `Update` method of the `HanaDataAdapter`.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
HanaCommand deleteCmd = new HanaCommand(
    "DELETE FROM DEMO.Departments WHERE DepartmentID >= 600", conn);
deleteCmd.ExecuteNonQuery();
HanaDataAdapter da = new HanaDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new HanaCommand(
    "SELECT * FROM DEMO.Departments ORDER By DepartmentID", conn );
da.InsertCommand = new HanaCommand(
    "INSERT INTO DEMO.Departments( DepartmentID, DepartmentName ) " +
    "VALUES( ?, ? )", conn );
da.InsertCommand.UpdatedRowSource =
    UpdateRowSource.None;
HanaParameter parm = new HanaParameter();
parm.HanaDbType = HanaDbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );
parm = new HanaParameter();
parm.HanaDbType = HanaDbType.VarChar;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );
DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );
DataRow row1 = dataTable.NewRow();
row1[0] = 600;
row1[1] = "Eastern Sales";
dataTable.Rows.Add( row1 );
DataRow row2 = dataTable.NewRow();
row2[0] = 700;
row2[1] = "Western Sales";
dataTable.Rows.Add( row2 );
rowCount = da.Update( dataTable );
dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

4.1.3.4.5.4 HanaDataAdapter: Delete Rows by Using Delete and Update

The `HanaDataAdapter` allows you to delete rows from a database table by using the `Delete` and `Update` methods.

Microsoft C# HanaDataAdapter Delete Example

This example shows how to use the `Update` method of `HanaDataAdapter` to delete rows from a table. The example adds two new rows to the Departments table and then fetches this table into a `DataTable` by using the `SelectCommand` property and the `Fill` method of the `HanaDataAdapter`. It then deletes some rows from the `DataTable` and updates the Departments table from the `DataTable` by using the `DeleteCommand` property and the `Update` method of the `HanaDataAdapter`.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
HanaCommand prepCmd = new HanaCommand("", conn);
prepCmd.CommandText =
    "DELETE FROM DEMO.Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO DEMO.Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO DEMO.Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();
HanaDataAdapter da = new HanaDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
da.SelectCommand = new HanaCommand(
    "SELECT * FROM DEMO.Departments ORDER By DepartmentID", conn);
da.DeleteCommand = new HanaCommand(
    "DELETE FROM DEMO.Departments WHERE DepartmentID = ?",
    conn);
da.DeleteCommand.UpdatedRowSource = UpdateRowSource.None;
HanaParameter parm = new HanaParameter();
parm.HanaDbType = HanaDbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.DeleteCommand.Parameters.Add(parm);
DataTable dataTable = new DataTable("Departments");
int rowCount = da.Fill(dataTable);
foreach (DataRow row in dataTable.Rows)
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row.Delete();
    }
}
rowCount = da.Update(dataTable);
dataTable.Clear();
rowCount = da.Fill(dataTable);
conn.Close();
dataGridView1.DataSource = dataTable;
```

4.1.3.4.5.5 HanaDataAdapter: Update Rows by Using Update

The `HanaDataAdapter` allows you to update rows in a database table by using the `Update` method.

Microsoft C# HanaDataAdapter Update Example

This example shows how to use the `Update` method of `HanaDataAdapter` to update rows in a table. The example adds two new rows to the Departments table and then fetches this table into a `DataTable` by using the `SelectCommand` property and the `Fill` method of the `HanaDataAdapter`. It then modifies some values in the `DataTable` and updates the Departments table from the `DataTable` by using the `UpdateCommand` property and the Update method of the `HanaDataAdapter`.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
HanaCommand prepCmd = new HanaCommand("", conn);
prepCmd.CommandText =
    "DELETE FROM DEMO.Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO DEMO.Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO DEMO.Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();
HanaDataAdapter da = new HanaDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new HanaCommand(
    "SELECT * FROM DEMO.Departments ORDER BY DepartmentID", conn );
da.UpdateCommand = new HanaCommand(
    "UPDATE DEMO.Departments SET DepartmentName = ? " +
    "WHERE DepartmentID = ?",
    conn );
da.UpdateCommand.UpdatedRowSource = UpdateRowSource.None;
HanaParameter parm = new HanaParameter();
parm.HanaDbType = HanaDbType.NVarChar;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.UpdateCommand.Parameters.Add( parm );
parm = new HanaParameter();
parm.HanaDbType = HanaDbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.UpdateCommand.Parameters.Add( parm );
DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );
foreach ( DataRow row in dataTable.Rows )
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row[1] = (string)row[1] + "_Updated";
    }
}
rowCount = da.Update( dataTable );
dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

4.1.3.4.6 Time Values (SAP HANA Data Provider for Microsoft ADO.NET)

The Microsoft .NET Framework does not have a `Time` structure. To fetch time values from an SAP HANA database server, use the `GetTimeSpan` method.

This method returns the data as a Microsoft .NET Framework `TimeSpan` object.

Microsoft C# TimeSpan Example

This example uses the `GetTimeSpan` method to return the current time as a `TimeSpan` value.

```
HanaConnection conn = new HanaConnection(
    "server=<hana-server:port>;UserID=<uid>;password=<pwd>" );
conn.Open();
HanaCommand cmd = new HanaCommand("SELECT CURRENT_TIME FROM DUMMY", conn);
HanaDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    TimeSpan time = reader.GetTimeSpan(0);
}
reader.Close();
conn.Close();
```

4.1.4 SAP HANA Data Provider for ADO.NET API Reference

The SAP HANA Data Provider for ADO.NET enables ADO.NET applications to connect to SAP HANA databases.

SAP HANA Data Provider for ADO.NET Relationship to .NET System.Data.Common

Many of the SAP HANA Data Provider for ADO.NET classes inherit from the standard .NET System.Data.Common classes. Further details about these classes can be found in the Microsoft .NET documentation.

SAP HANA Data Provider for ADO.NET	Microsoft System.Data.Common
HanaCommand Class	DbCommand Class
HanaCommandBuilder Class	DbCommandBuilder Class
HanaConnection Class	DbConnection Class

SAP HANA Data Provider for ADO.NET	Microsoft System.Data.Common
HanaConnectionStringBuilder Class	DbConnectionStringBuilder Class
HanaDataAdapter Class	DbDataAdapter Class
HanaDataReader Class	DbDataReader Class
HanaDataSourceEnumerator Class	DbDataSourceEnumerator Class
HanaParameter Class	DbParameter Class
HanaParameterCollection Class	DbParameterCollection Class
HanaTransaction Class	DbTransaction Class

SAP HANA Data Provider for ADO.NET Namespace

The SAP HANA Data Provider for ADO.NET is contained within two namespaces:

- `Sap.Data.Hana`
- `Sap.Hana.Server`

4.1.4.1 HanaBulkCopy class

Efficiently bulk loads an SAP HANA table with data from another source.

Syntax

Visual Basic

```
Public NotInheritable Class HanaBulkCopy Implements
System.IDisposable
```

C#

```
public sealed class HanaBulkCopy : System.IDisposable
```

Members

All members of HanaBulkCopy, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaBulkCopy [page 112]	Initializes a HanaBulkCopy object.

Methods

Modifier and Type	Method	Description
public void	Close() [page 115]	Closes the HanaBulkCopy instance.
public void	Dispose() [page 115]	Disposes of the HanaBulkCopy instance.
public void	WriteToServer [page 116]	Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.
public Task	WriteToServerAsync [page 119]	Copies all rows from the supplied System.Data.DataRow array to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.

Properties

Modifier and Type	Property	Description
public int	BatchSize [page 127]	Gets or sets the number of rows in each batch.
public int	BulkCopyTimeout [page 128]	Gets or sets the number of seconds for the operation to complete before it times out.
public HanaBulkCopyColumnMapping-Collection	ColumnMappings [page 128]	Returns a collection of HanaBulkCopy-ColumnMapping items.
public string	DestinationTableName [page 129]	Gets or sets the name of the destination table on the SAP HANA database server.
public int	NotifyAfter [page 130]	Gets or sets the number of rows to be processed before generating a notification event.

Events

Modifier and Type	Event	Description
public HanaRowsCopiedEventHandler	HanaRowsCopied [page 130]	This event occurs every time the number of rows specified by the NotifyAfter property have been processed.

Remarks

Implements: System.IDisposable

4.1.4.1.1 HanaBulkCopy constructor

Initializes a HanaBulkCopy object.

Overload list

Modifier and Type	Overload name	Description
public	HanaBulkCopy(HanaConnection) [page 112]	Initializes a HanaBulkCopy object.
public	HanaBulkCopy(HanaConnection, HanaBulkCopyOptions, HanaTransaction) [page 113]	Initializes a HanaBulkCopy object.
public	HanaBulkCopy(string) [page 113]	Initializes a HanaBulkCopy object.
public	HanaBulkCopy(string, HanaBulkCopyOptions) [page 114]	Initializes a HanaBulkCopy object.

4.1.4.1.1.1 HanaBulkCopy(HanaConnection) constructor

Initializes a HanaBulkCopy object.

Syntax

Visual Basic

```
Public Sub HanaBulkCopy (ByVal connection As HanaConnection)
```

C#

```
public HanaBulkCopy (HanaConnection connection)
```

Parameters

connection

A HanaConnection object that is used to perform the bulk-copy operation. If the connection is not open, then an exception is thrown in WriteToServer.

4.1.4.1.1.2 HanaBulkCopy(HanaConnection, HanaBulkCopyOptions, HanaTransaction) constructor

Initializes a HanaBulkCopy object.

Syntax

Visual Basic

```
Public Sub HanaBulkCopy (
    ByVal connection As HanaConnection,
    ByVal copyOptions As HanaBulkCopyOptions,
    ByVal externalTransaction As HanaTransaction
)
```

C#

```
public HanaBulkCopy (
    HanaConnection connection,
    HanaBulkCopyOptions copyOptions,
    HanaTransaction externalTransaction
)
```

Parameters

connection

A HanaConnection object that is used to perform the bulk-copy operation. If the connection is not open, then an exception is thrown in WriteToServer.

copyOptions

A combination of values from the HanaBulkCopyOptions enumeration that determines which data source rows are copied to the destination table.

externalTransaction

An existing HanaTransaction instance under which the bulk copy will occur. If externalTransaction is not NULL, then the bulk-copy operation is done within it. It is an error to specify both an external transaction and the UseInternalTransaction option.

4.1.4.1.1.3 HanaBulkCopy(string) constructor

Initializes a HanaBulkCopy object.

Syntax

Visual Basic

```
Public Sub HanaBulkCopy (ByVal connectionString As String)
```

C#

```
public HanaBulkCopy (string connectionString)
```

Parameters

connectionString

The string defining the connection that is opened for use by the HanaBulkCopy instance. A connection string is a semicolon-separated list of keyword=value pairs.

Remarks

This syntax opens a connection during WriteToServer using connectionString. The connection is closed at the end of WriteToServer.

4.1.4.1.1.4 HanaBulkCopy(string, HanaBulkCopyOptions) constructor

Initializes a HanaBulkCopy object.

Syntax

Visual Basic

```
Public Sub HanaBulkCopy (
    ByVal connectionString As String,
    ByVal copyOptions As HanaBulkCopyOptions
)
```

C#

```
public HanaBulkCopy (
    string connectionString,
    HanaBulkCopyOptions copyOptions
)
```

Parameters

connectionString

The string defining the connection that is opened for use by the HanaBulkCopy instance. A connection string is a semicolon-separated list of keyword=value pairs.

copyOptions

A combination of values from the HanaBulkCopyOptions enumeration that determines which data source rows are copied to the destination table.

Remarks

This syntax opens a connection during WriteToServer using connectionString. The connection is closed at the end of WriteToServer. The copyOptions parameter has the effects described above.

4.1.4.1.2 Close() method

Closes the HanaBulkCopy instance.

Syntax

Visual Basic

```
Public Sub Close ()
```

C#

```
public void Close ()
```

4.1.4.1.3 Dispose() method

Disposes of the HanaBulkCopy instance.

Syntax

Visual Basic

```
Public Sub Dispose ()
```

C#

```
public void Dispose ()
```

4.1.4.1.4 WriteToServer method

Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.

Overload list

Modifier and Type	Overload name	Description
public void	WriteToServer(DataRow[]) [page 116]	Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.
public void	WriteToServer(DataTable) [page 117]	Copies all rows in the supplied System.Data.DataTable to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.
public void	WriteToServer(DataTable, DataRowState) [page 118]	Copies all rows in the supplied System.Data.DataTable with the specified row state to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.
public void	WriteToServer(IDataReader) [page 118]	Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.

4.1.4.1.4.1 WriteToServer(DataRow[]) method

Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.

Syntax

Visual Basic

```
Public Sub WriteToServer (ByVal rows As DataRow())
```

C#

```
public void WriteToServer (DataRow[] rows)
```

Parameters

rows

An array of System.Data.DataRow objects that is copied to the destination table.

Related Information

DestinationTableName property [page 129]

4.1.4.1.4.2 WriteToServer(DataTable) method

Copies all rows in the supplied System.Data.DataTable to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.

Syntax

Visual Basic

```
Public Sub WriteToServer (ByVal table As DataTable)
```

C#

```
public void WriteToServer (DataTable table)
```

Parameters

table

A System.Data.DataTable whose rows are copied to the destination table.

Related Information

DestinationTableName property [page 129]

4.1.4.1.4.3 WriteToServer(DataTable, DataRowState) method

Copies all rows in the supplied System.Data.DataTable with the specified row state to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.

Syntax

Visual Basic

```
Public Sub WriteToServer (
    ByVal table As DataTable,
    ByVal rowState As DataRowState
)
```

C#

```
public void WriteToServer (
    DataTable table,
    DataRowState rowState
)
```

Parameters

table

A System.Data.DataTable whose rows are copied to the destination table.

rowState

A value from the System.Data.DataRowState enumeration. Only rows matching the row state are copied to the destination.

Remarks

Only those rows matching the row state are copied.

4.1.4.1.4.4 WriteToServer(IDataReader) method

Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the DestinationTableName property of the HanaBulkCopy object.

Syntax

Visual Basic

```
Public Sub WriteToServer (ByVal reader As IDataReader)
```

C#

```
public void WriteToServer (IDataReader reader)
```

Parameters

reader

A System.Data.IDataReader whose rows are copied to the destination table.

Related Information

DestinationTableName property [page 129]

4.1.4.1.5 WriteToServerAsync method

Copies all rows from the supplied System.Data.DataRow array to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.

Overload list

Modifier and Type	Overload name	Description
public Task	WriteToServerAsync(DataRow[]) [page 120]	Copies all rows from the supplied System.Data.DataRow array to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.
public Task	WriteToServerAsync(DataRow[], CancellationToken) [page 121]	Copies all rows from the supplied System.Data.DataRow array to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.
public Task	WriteToServerAsync(DataTable) [page 122]	Copies all rows in the supplied System.Data.DataTable to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.
public Task	WriteToServerAsync(DataTable, CancellationToken) [page 123]	Copies all rows in the supplied System.Data.DataTable to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object. The cancellation token can be used to request that the operation be abandoned before the command timeout elapses.

Modifier and Type	Overload name	Description
public Task	WriteToServerAsync(DataTable, DataRowState) [page 124]	Copies only rows that match the supplied row state in the supplied System.Data.DataTable to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.
public Task	WriteToServerAsync(DataTable, DataRowState, CancellationToken) [page 124]	Copies only rows that match the supplied row state in the supplied System.Data.DataTable to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object. The cancellation token can be used to request that the operation be abandoned before the command timeout elapses.
public Task	WriteToServerAsync(IDataReader) [page 126]	Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.
public Task	WriteToServerAsync(IDataReader, CancellationToken) [page 126]	Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the HanaClient.SqlBulkCopy.DestinationTableName property of the HanaClient.SqlBulkCopy object. The cancellation token can be used to request that the operation be abandoned before the command timeout elapses.

4.1.4.1.5.1 WriteToServerAsync(DataRow[]) method

Copies all rows from the supplied System.Data.DataRow array to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.

Syntax

Visual Basic

```
Public Function WriteToServerAsync (ByVal rows As DataRow()) As
Task
```

C#

```
public Task WriteToServerAsync (DataRow[] rows)
```

Parameters

rows

An array of System.Data.DataRow objects that are copied to the destination table.

4.1.4.1.5.2 WriteToServerAsync(DataRow[], CancellationToken) method

Copies all rows from the supplied System.Data.DataRow array to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.

Syntax

Visual Basic

```
Public Function WriteToServerAsync (
    ByVal rows As DataRow(),
    ByVal cancellationToken As CancellationToken
) As Task
```

C#

```
public Task WriteToServerAsync (
    DataRow[] rows,
    CancellationToken cancellationToken
)
```

Parameters

rows

An array of System.Data.DataRow objects that are copied to the destination table.

cancellationToken

The cancellation instruction. A P:System.Threading.CancellationToken.None value in this parameter makes this method equivalent to HanaBulkCopy.WriteToServerAsync(System.Data.DataTable).

Returns

A task representing the asynchronous operation.

Exceptions

HanaException class

Returned in the task object, any error returned by SAP HANA that occurred while copying data.

Remarks

The cancellation token can be used to request that the operation be abandoned before the command timeout elapses. Exceptions are reported via the returned Task object.

4.1.4.1.5.3 WriteToServerAsync(DataTable) method

Copies all rows in the supplied System.Data.DataTable to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.

Syntax

Visual Basic

```
Public Function WriteToServerAsync (ByVal table As DataTable) As
Task
```

C#

```
public Task WriteToServerAsync (DataTable table)
```

Parameters

table

A System.Data.DataTable whose rows are copied to the destination table.

4.1.4.1.5.4 WriteToServerAsync(DataTable, CancellationToken) method

Copies only rows in the supplied System.Data.DataTable to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object. The cancellation token can be used to request that the operation be abandoned before the command timeout elapses.

Syntax

Visual Basic

```
Public Function WriteToServerAsync (
    ByVal table As DataTable,
    ByVal cancellationToken As CancellationToken
) As Task
```

C#

```
public Task WriteToServerAsync (
    DataTable table,
    CancellationToken cancellationToken
)
```

Parameters

table

A System.Data.DataTable whose rows are copied to the destination table.

cancellationToken

The cancellation instruction. A P:System.Threading.CancellationToken.None value in this parameter makes this method equivalent to HanaBulkCopy.WriteToServerAsync(System.Data.DataTable).

Returns

A task representing the asynchronous operation.

Exceptions

HanaException class

Returned in the task object, any error returned by the SAP HANA database server that occurred while copying data.

Remarks

Exceptions are reported via the returned Task object.

4.1.4.1.5.5 WriteToServerAsync(DataTable, DataRowState) method

Copies only rows that match the supplied row state in the supplied System.Data.DataTable to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.

Syntax

Visual Basic

```
Public Function WriteToServerAsync (
    ByVal table As DataTable,
    ByVal rowState As DataRowState
) As Task
```

C#

```
public Task WriteToServerAsync (
    DataTable table,
    DataRowState rowState
)
```

Parameters

table

A System.Data.DataTable whose rows are copied to the destination table.

rowState

A value from the System.Data.DataRowState enumeration. Only rows matching the row state are copied to the destination.

4.1.4.1.5.6 WriteToServerAsync(DataTable, DataRowState, CancellationToken) method

Copies only rows that match the supplied row state in the supplied System.Data.DataTable to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object. The

cancellation token can be used to request that the operation be abandoned before the command timeout elapses.

Syntax

Visual Basic

```
Public Function WriteToServerAsync (
    ByVal table As DataTable,
    ByVal rowState As DataRowState,
    ByVal cancellationToken As CancellationToken
) As Task
```

C#

```
public Task WriteToServerAsync (
    DataTable table,
    DataRowState rowState,
    CancellationToken cancellationToken
)
```

Parameters

table

A System.Data.DataTable whose rows are copied to the destination table.

rowState

A value from the System.Data.DataRowState enumeration. Only rows matching the row state are copied to the destination.

cancellationToken

The cancellation instruction. A P:System.Threading.CancellationToken.None value in this parameter makes this method equivalent to HanaBulkCopy.WriteToServerAsync(System.Data.DataTable, System.Data.DataRowState).

Returns

A task representing the asynchronous operation.

Exceptions

HanaException class

Returned in the task object, any error returned by the SAP HANA database server that occurred while copying data.

Remarks

Exceptions are reported via the returned Task object.

4.1.4.1.5.7 WriteToServerAsync(IDataReader) method

Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the HanaBulkCopy.DestinationTableName property of the HanaBulkCopy object.

Syntax

Visual Basic

```
Public Function WriteToServerAsync (ByVal reader As IDataReader)
As Task
```

C#

```
public Task WriteToServerAsync (IDataReader reader)
```

Parameters

reader

A System.Data.IDataReader whose rows are copied to the destination table.

4.1.4.1.5.8 WriteToServerAsync(IDataReader, CancellationToken) method

Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the HanaClient.SqlBulkCopy.DestinationTableName property of the HanaClient.SqlBulkCopy object. The cancellation token can be used to request that the operation be abandoned before the command timeout elapses.

Syntax

Visual Basic

```
Public Function WriteToServerAsync (
    ByVal reader As IDataReader,
    ByVal cancellationToken As CancellationToken
) As Task
```

C#

```
public Task WriteToServerAsync (
    IDataReader reader,
    CancellationToken cancellationToken
)
```

Parameters

reader

A System.Data.IDataReader whose row are copied to the destination table.

cancellationToken

The cancellation instruction. A P:System.Threading.CancellationToken.None value in this parameter makes this method equivalent to HanaBulkCopy.WriteToServerAsync(System.Data.IDataReader).

Returns

A task representing the asynchronous operation.

Exceptions

HanaException class

Returned in the task object, any error returned by the SAP HANA database server that occurred while copying data.

Remarks

Exceptions are reported via the returned Task object.

4.1.4.1.6 BatchSize property

Gets or sets the number of rows in each batch.

Syntax

Visual Basic

```
Public Property BatchSize As Integer
```

C#

```
public int BatchSize {get;set;}
```

Remarks

At the end of each batch, the rows in the batch are sent to the SAP HANA database server.

The number of rows in each batch. The default is 0.

Setting this property to zero causes all the rows to be sent in one batch.

Setting this property to a value less than zero is an error.

If this value is changed while a batch is in progress, then the current batch completes and any further batches use the new value.

4.1.4.1.7 BulkCopyTimeout property

Gets or sets the number of seconds for the operation to complete before it times out.

Syntax

Visual Basic

```
Public Property BulkCopyTimeout As Integer
```

C#

```
public int BulkCopyTimeout {get;set;}
```

Remarks

The default value is 30 seconds.

A value of zero indicates no limit. This value should be avoided because it may cause an indefinite wait.

If the operation times out, then all rows in the current transaction are rolled back and a HanaException is raised.

Setting this property to a value less than zero is an error.

4.1.4.1.8 ColumnMappings property

Returns a collection of HanaBulkCopyColumnMapping items.

Syntax

Visual Basic

```
Public ReadOnly Property ColumnMappings As
HanaBulkCopyColumnMappingCollection
```

C#

```
public HanaBulkCopyColumnMappingCollection ColumnMappings {get;}
```

Remarks

Column mappings define the relationships between columns in the data source and columns in the destination.

By default, it is an empty collection.

The property cannot be modified while WriteToServer is executing.

If ColumnMappings is empty when WriteToServer is executed, then the first column in the source is mapped to the first column in the destination, the second to the second, and so on. This behavior takes place as long as the column types are convertible, there are at least as many destination columns as source columns, and any extra destination columns are nullable.

4.1.4.1.9 DestinationTableName property

Gets or sets the name of the destination table on the SAP HANA database server.

Syntax

Visual Basic

```
Public Property DestinationTableName As String
```

C#

```
public string DestinationTableName {get;set;}
```

Remarks

The default value is a null reference. In Visual Basic it is Nothing.

If the value is changed while WriteToServer is executing, then the change has no effect.

If the value has not been set before a call to WriteToServer, then an InvalidOperationException is raised.

It is an error to set the value to NULL or the empty string.

4.1.4.1.10 NotifyAfter property

Gets or sets the number of rows to be processed before generating a notification event.

Syntax

Visual Basic

```
Public Property NotifyAfter As Integer
```

C#

```
public int NotifyAfter {get;set;}
```

Remarks

Zero is returned if the property has not been set.

Changes made to NotifyAfter, while executing WriteToServer, do not take effect until after the next notification.

Setting this property to a value less than zero is an error.

The values of NotifyAfter and BulkCopyTimeout are mutually exclusive, so the event can fire even if no rows have been sent to the database or committed.

Related Information

BulkCopyTimeout property [page 128]

4.1.4.1.11 HanaRowsCopied event

This event occurs every time the number of rows specified by the NotifyAfter property have been processed.

Syntax

Visual Basic

```
Public Event HanaRowsCopied  As HanaRowsCopiedEventHandler
```

C#

```
public HanaRowsCopiedEventHandler HanaRowsCopied;
```

Remarks

The receipt of a HanaRowsCopied event does not imply that any rows have been sent to the SAP HANA database server or committed. You cannot call the Close method from this event.

Related Information

NotifyAfter property [page 130]

4.1.4.2 HanaBulkCopyColumnMapping class

Defines the mapping between a column in a HanaBulkCopy instance's data source and a column in the instance's destination table.

Syntax

Visual Basic

```
Public NotInheritable Class HanaBulkCopyColumnMapping
```

C#

```
public sealed class HanaBulkCopyColumnMapping
```

Members

All members of HanaBulkCopyColumnMapping, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaBulkCopyColumnMapping [page 132]	Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

Properties

Modifier and Type	Property	Description
public string	DestinationColumn [page 135]	Gets or sets the name of the column in the destination database table being mapped to.

Modifier and Type	Property	Description
public int	DestinationOrdinal [page 136]	Gets or sets the ordinal value of the column in the destination table being mapped to.
public string	SourceColumn [page 137]	Gets or sets the name of the column being mapped in the data source.
public int	SourceOrdinal [page 138]	Gets or sets ordinal position of the source column within the data source.

4.1.4.2.1 HanaBulkCopyColumnMapping constructor

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

Overload list

Modifier and Type	Overload name	Description
public	HanaBulkCopyColumnMapping() [page 133]	Creates a new column mapping, using column ordinals or names to refer to source and destination columns.
public	HanaBulkCopyColumnMapping(int, int) [page 133]	Creates a new column mapping, using column ordinals to refer to source and destination columns.
public	HanaBulkCopyColumnMapping(int, string) [page 134]	Creates a new column mapping, using a column ordinal to refer to the source column and a column name to refer to the destination column.
public	HanaBulkCopyColumnMapping(string, int) [page 134]	Creates a new column mapping, using a column name to refer to the source column and a column ordinal to refer to the destination column.
public	HanaBulkCopyColumnMapping(string, string) [page 135]	Creates a new column mapping, using column names to refer to source and destination columns.

4.1.4.2.1.1 HanaBulkCopyColumnMapping() constructor

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

Syntax

Visual Basic

```
Public Sub HanaBulkCopyColumnMapping ()
```

C#

```
public HanaBulkCopyColumnMapping ()
```

4.1.4.2.1.2 HanaBulkCopyColumnMapping(int, int) constructor

Creates a new column mapping, using column ordinals to refer to source and destination columns.

Syntax

Visual Basic

```
Public Sub HanaBulkCopyColumnMapping (
    ByVal sourceColumnOrdinal As Integer,
    ByVal destinationColumnOrdinal As Integer
)
```

C#

```
public HanaBulkCopyColumnMapping (
    int sourceColumnOrdinal,
    int destinationColumnOrdinal
)
```

Parameters

sourceColumnOrdinal

The ordinal position of the source column within the data source. The first column in a data source has ordinal position zero.

destinationColumnOrdinal

The ordinal position of the destination column within the destination table. The first column in a table has ordinal position zero.

4.1.4.2.1.3 HanaBulkCopyColumnMapping(int, string) constructor

Creates a new column mapping, using a column ordinal to refer to the source column and a column name to refer to the destination column.

Syntax

Visual Basic

```
Public Sub HanaBulkCopyColumnMapping (
    ByVal sourceColumnOrdinal As Integer,
    ByVal destinationColumn As String
)
```

C#

```
public HanaBulkCopyColumnMapping (
    int sourceColumnOrdinal,
    string destinationColumn
)
```

Parameters

sourceColumnOrdinal

The ordinal position of the source column within the data source. The first column in a data source has ordinal position zero.

destinationColumn

The name of the destination column within the destination table.

4.1.4.2.1.4 HanaBulkCopyColumnMapping(string, int) constructor

Creates a new column mapping, using a column name to refer to the source column and a column ordinal to refer to the destination column.

Syntax

Visual Basic

```
Public Sub HanaBulkCopyColumnMapping (
    ByVal sourceColumn As String,
    ByVal destinationColumnOrdinal As Integer
)
```

C#

```
public HanaBulkCopyColumnMapping (
    string sourceColumn,
    int destinationColumnOrdinal
)
```

Parameters

sourceColumn

The name of the source column within the data source.

destinationColumnOrdinal

The ordinal position of the destination column within the destination table. The first column in a table has ordinal position zero.

4.1.4.2.1.5 HanaBulkCopyColumnMapping(string, string) constructor

Creates a new column mapping, using column names to refer to source and destination columns.

Syntax

Visual Basic

```
Public Sub HanaBulkCopyColumnMapping (
    ByVal sourceColumn As String,
    ByVal destinationColumn As String
)
```

C#

```
public HanaBulkCopyColumnMapping (
    string sourceColumn,
    string destinationColumn
)
```

Parameters

sourceColumn

The name of the source column within the data source.

destinationColumn

The name of the destination column within the destination table.

4.1.4.2.2 DestinationColumn property

Gets or sets the name of the column in the destination database table being mapped to.

Syntax

Visual Basic

```
Public Property DestinationColumn As String
```

C#

```
public string DestinationColumn {get;set;}
```

Remarks

A string specifying the name of the column in the destination table or a null reference (Nothing in Visual Basic) if the DestinationOrdinal property has priority.

The DestinationColumn property and DestinationOrdinal property are mutually exclusive. The most recently set value takes priority.

Setting the DestinationColumn property causes the DestinationOrdinal property to be set to -1. Setting the DestinationOrdinal property causes the DestinationColumn property to be set to a null reference (Nothing in Visual Basic).

It is an error to set DestinationColumn to null or the empty string.

Related Information

DestinationOrdinal property [page 136]

4.1.4.2.3 DestinationOrdinal property

Gets or sets the ordinal value of the column in the destination table being mapped to.

Syntax

Visual Basic

```
Public Property DestinationOrdinal As Integer
```

C#

```
public int DestinationOrdinal {get;set;}
```

Remarks

An integer specifying the ordinal of the column being mapped to in the destination table or -1 if the property is not set.

The DestinationColumn property and DestinationOrdinal property are mutually exclusive. The most recently set value takes priority.

Setting the DestinationColumn property causes the DestinationOrdinal property to be set to -1. Setting the DestinationOrdinal property causes the DestinationColumn property to be set to a null reference (Nothing in Visual Basic).

Related Information

DestinationColumn property [page 135]

4.1.4.2.4 SourceColumn property

Gets or sets the name of the column being mapped in the data source.

Syntax

Visual Basic

```
Public Property SourceColumn As String
```

C#

```
public string SourceColumn {get;set;}
```

Remarks

A string specifying the name of the column in the data source or a null reference (Nothing in Visual Basic) if the SourceOrdinal property has priority.

The SourceColumn property and SourceOrdinal property are mutually exclusive. The most recently set value takes priority.

Setting the SourceColumn property causes the SourceOrdinal property to be set to -1. Setting the SourceOrdinal property causes the SourceColumn property to be set to a null reference (Nothing in Visual Basic).

It is an error to set SourceColumn to null or the empty string.

Related Information

SourceOrdinal property [page 138]

4.1.4.2.5 SourceOrdinal property

Gets or sets ordinal position of the source column within the data source.

Syntax

Visual Basic

```
Public Property SourceOrdinal As Integer
```

C#

```
public int SourceOrdinal {get;set;}
```

Remarks

An integer specifying the ordinal of the column in the data source or -1 if the property is not set.

The SourceColumn property and SourceOrdinal property are mutually exclusive. The most recently set value takes priority.

Setting the SourceColumn property causes the SourceOrdinal property to be set to -1. Setting the SourceOrdinal property causes the SourceColumn property to be set to a null reference (Nothing in Visual Basic).

Related Information

SourceColumn property [page 137]

4.1.4.3 HanaBulkCopyColumnMappingCollection class

A collection of HanaBulkCopyColumnMapping objects that inherits from System.Collections.CollectionBase.

Syntax

Visual Basic

```
Public NotInheritable Class HanaBulkCopyColumnMappingCollection
Inherits System.Collections.CollectionBase
```

C#

```
public sealed class HanaBulkCopyColumnMappingCollection :
System.Collections.CollectionBase
```

Members

All members of HanaBulkCopyColumnMappingCollection, including inherited members.

Methods

Modifier and Type	Method	Description
public HanaBulkCopyColumnMapping	Add [page 140]	Adds the specified HanaBulkCopyColumnMapping object to the collection.
public bool	Contains(HanaBulkCopyColumnMapping) [page 144]	Gets a value indicating whether a specified HanaBulkCopyColumnMapping object exists in the collection.
public void	CopyTo(HanaBulkCopyColumnMapping[], int) [page 144]	Copies the elements of the HanaBulkCopyColumnMappingCollection to an array of HanaBulkCopyColumnMapping items, starting at a particular index.
public int	IndexOf(HanaBulkCopyColumnMapping) [page 145]	Gets or sets the index of the specified HanaBulkCopyColumnMapping object within the collection.
public void	Remove(HanaBulkCopyColumnMapping) [page 146]	Removes the specified HanaBulkCopyColumnMapping element from the HanaBulkCopyColumnMappingCollection.
public new void	RemoveAt(int) [page 146]	Removes the mapping at the specified index from the collection.

Properties

Modifier and Type	Property	Description
public HanaBulkCopyColumnMapping	this[int index] [page 147]	Gets the HanaBulkCopyColumnMapping object at the specified index.

Remarks

Implements: ICollection, IEnumerable, IList

4.1.4.3.1 Add method

Adds the specified HanaBulkCopyColumnMapping object to the collection.

Overload list

Modifier and Type	Overload name	Description
public HanaBulkCopyColumnMapping	Add(HanaBulkCopyColumnMapping) [page 140]	Adds the specified HanaBulkCopyColumnMapping object to the collection.
public HanaBulkCopyColumnMapping	Add(int, int) [page 141]	Creates a new HanaBulkCopyColumnMapping object using ordinals to specify both source and destination columns, and adds the mapping to the collection.
public HanaBulkCopyColumnMapping	Add(int, string) [page 142]	Creates a new HanaBulkCopyColumnMapping object using a column ordinal to refer to the source column and a column name to refer to the destination column, and adds mapping to the collection.
public HanaBulkCopyColumnMapping	Add(string, int) [page 142]	Creates a new HanaBulkCopyColumnMapping object using a column name to refer to the source column and a column ordinal to refer to the destination the column, and adds the mapping to the collection.
public HanaBulkCopyColumnMapping	Add(string, string) [page 143]	Creates a new HanaBulkCopyColumnMapping object using column names to specify both source and destination columns, and adds the mapping to the collection.

4.1.4.3.1.1 Add(HanaBulkCopyColumnMapping) method

Adds the specified HanaBulkCopyColumnMapping object to the collection.

Syntax

Visual Basic

```
Public Function Add (ByVal bulkCopyColumnMapping As
HanaBulkCopyColumnMapping) As HanaBulkCopyColumnMapping
```

C#

```
public HanaBulkCopyColumnMapping Add (HanaBulkCopyColumnMapping
bulkCopyColumnMapping)
```

Parameters

bulkCopyColumnMapping

The HanaBulkCopyColumnMapping object that describes the mapping to be added to the collection.

Related Information

HanaBulkCopyColumnMapping class [page 131]

4.1.4.3.1.2 Add(int, int) method

Creates a new HanaBulkCopyColumnMapping object using ordinals to specify both source and destination columns, and adds the mapping to the collection.

Syntax

Visual Basic

```
Public Function Add (
    ByVal sourceColumnOrdinal As Integer,
    ByVal destinationColumnOrdinal As Integer
) As HanaBulkCopyColumnMapping
```

C#

```
public HanaBulkCopyColumnMapping Add (
    int sourceColumnOrdinal,
    int destinationColumnOrdinal
)
```

Parameters

sourceColumnOrdinal

The ordinal position of the source column within the data source.

destinationColumnOrdinal

The ordinal position of the destination column within the destination table.

4.1.4.3.1.3 Add(int, string) method

Creates a new HanaBulkCopyColumnMapping object using a column ordinal to refer to the source column and a column name to refer to the destination column, and adds mapping to the collection.

Syntax

Visual Basic

```
Public Function Add (
    ByVal sourceColumnOrdinal As Integer,
    ByVal destinationColumn As String
) As HanaBulkCopyColumnMapping
```

C#

```
public HanaBulkCopyColumnMapping Add (
    int sourceColumnOrdinal,
    string destinationColumn
)
```

Parameters

sourceColumnOrdinal

The ordinal position of the source column within the data source.

destinationColumn

The name of the destination column within the destination table.

4.1.4.3.1.4 Add(string, int) method

Creates a new HanaBulkCopyColumnMapping object using a column name to refer to the source column and a column ordinal to refer to the destination the column, and adds the mapping to the collection.

Syntax

Visual Basic

```
Public Function Add (
    ByVal sourceColumn As String,
    ByVal destinationColumnOrdinal As Integer
) As HanaBulkCopyColumnMapping
```

C#

```
public HanaBulkCopyColumnMapping Add (
    string sourceColumn,
    int destinationColumnOrdinal
)
```

Parameters

sourceColumn

The name of the source column within the data source.

destinationColumnOrdinal

The ordinal position of the destination column within the destination table.

Remarks

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

4.1.4.3.1.5 Add(string, string) method

Creates a new HanaBulkCopyColumnMapping object using column names to specify both source and destination columns, and adds the mapping to the collection.

Syntax

Visual Basic

```
Public Function Add (
    ByVal sourceColumn As String,
    ByVal destinationColumn As String
) As HanaBulkCopyColumnMapping
```

C#

```
public HanaBulkCopyColumnMapping Add (
    string sourceColumn,
    string destinationColumn
)
```

Parameters

sourceColumn

The name of the source column within the data source.

destinationColumn

The name of the destination column within the destination table.

4.1.4.3.2 Contains(HanaBulkCopyColumnMapping) method

Gets a value indicating whether a specified HanaBulkCopyColumnMapping object exists in the collection.

Syntax

Visual Basic

```
Public Function Contains (ByVal value As
HanaBulkCopyColumnMapping) As Boolean
```

C#

```
public bool Contains (HanaBulkCopyColumnMapping value)
```

Parameters

value

A valid HanaBulkCopyColumnMapping object.

Returns

True if the specified mapping exists in the collection; false otherwise.

4.1.4.3.3 CopyTo(HanaBulkCopyColumnMapping[], int) method

Copies the elements of the HanaBulkCopyColumnMappingCollection to an array of HanaBulkCopyColumnMapping items, starting at a particular index.

Syntax

Visual Basic

```
Public Sub CopyTo (
    ByVal array As HanaBulkCopyColumnMapping(),
    ByVal index As Integer
)
```

C#

```
public void CopyTo (
    HanaBulkCopyColumnMapping[] array,
    int index
)
```

Parameters

array

The one-dimensional HanaBulkCopyColumnMapping array that is the destination of the elements copied from HanaBulkCopyColumnMappingCollection. The array must have zero-based indexing.

index

The zero-based index in the array at which copying begins.

4.1.4.3.4 IndexOf(HanaBulkCopyColumnMapping) method

Gets or sets the index of the specified HanaBulkCopyColumnMapping object within the collection.

Syntax

Visual Basic

```
Public Function IndexOf (ByVal value As HanaBulkCopyColumnMapping)
As Integer
```

C#

```
public int IndexOf (HanaBulkCopyColumnMapping value)
```

Parameters

value

The HanaBulkCopyColumnMapping object to search for.

Returns

The zero-based index of the column mapping is returned, or -1 is returned if the column mapping is not found in the collection.

4.1.4.3.5 Remove(HanaBulkCopyColumnMapping) method

Removes the specified HanaBulkCopyColumnMapping element from the HanaBulkCopyColumnMappingCollection.

Syntax

Visual Basic

```
Public Sub Remove (ByVal value As HanaBulkCopyColumnMapping)
```

C#

```
public void Remove (HanaBulkCopyColumnMapping value)
```

Parameters

value

The HanaBulkCopyColumnMapping object to be removed from the collection.

4.1.4.3.6 RemoveAt(int) method

Removes the mapping at the specified index from the collection.

Syntax

Visual Basic

```
Public Shadows Sub RemoveAt (ByVal index As Integer)
```

C#

```
public new void RemoveAt (int index)
```

Parameters

index

The zero-based index of the HanaBulkCopyColumnMapping object to be removed from the collection.

4.1.4.3.7 this[int index] property

Gets the HanaBulkCopyColumnMapping object at the specified index.

Syntax

Visual Basic

```
Public ReadOnly Property Item (ByVal index As Integer) As
HanaBulkCopyColumnMapping
```

C#

```
public HanaBulkCopyColumnMapping this[int index] {get;}
```

Returns

A HanaBulkCopyColumnMapping object is returned.

4.1.4.4 HanaCommand class

A SQL statement or stored procedure that is executed against an SAP HANA database.

Syntax

Visual Basic

```
Public NotInheritable Class HanaCommand Inherits
System.Data.Common.DbCommand Implements System.ICloneable
```

C#

```
public sealed class HanaCommand : System.Data.Common.DbCommand,
System.ICloneable
```

Members

All members of HanaCommand, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaCommand [page 150]	Initializes a HanaCommand object.

Methods

Modifier and Type	Method	Description
public IAsyncResult	BeginExecuteNonQuery [page 152]	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand.
public IAsyncResult	BeginExecuteReader [page 155]	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, and retrieves one or more result sets from the SAP HANA database server.
public override void	Cancel() [page 160]	Cancels the execution of a HanaCommand object.
protected override DbParameter	CreateDbParameter() [page 161]	Creates a new instance of a System.Data.Common.DbParameter object.
public new HanaParameter	CreateParameter() [page 161]	Provides a HanaParameter object for supplying parameters to HanaCommand objects.
protected override void	Dispose(bool) [page 162]	Frees the resources associated with the object.
public unsafe int	EndExecuteNonQuery(IAsyncResult) [page 162]	Finishes asynchronous execution of a SQL statement or stored procedure.
public unsafe HanaDataReader	EndExecuteReader(IAsyncResult) [page 165]	Finishes asynchronous execution of a SQL statement or stored procedure, returning the requested HanaDataReader.
protected override DbDataReader	ExecuteDbDataReader(CommandBehavior) [page 167]	Executes the command text against the connection.
public override unsafe int	ExecuteNonQuery() [page 168]	Executes a statement that does not return a result set, such as an INSERT, UPDATE, DELETE, or data definition statement.
public new HanaDataReader	ExecuteReader [page 169]	Executes a SQL statement that returns a result set.
public new Task< HanaDataReader >	ExecuteReaderAsync [page 171]	An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.
public override object	ExecuteScalar() [page 175]	Executes a statement that returns a single value.
public override void	Prepare() [page 176]	Prepares or compiles the HanaCommand on the data source.
public void	ResetCommandTimeout() [page 177]	Resets the CommandTimeout property to its default value of 30 seconds.

Properties

Modifier and Type	Property	Description
public override string	CommandText [page 177]	Gets or sets the text of a SQL statement or stored procedure.
public override int	CommandTimeout [page 178]	This feature is not supported by the data provider.
public override CommandType	CommandType [page 178]	Gets or sets the type of command represented by a HanaCommand.
public new HanaConnection	Connection [page 179]	Gets or sets the connection object to which the HanaCommand object applies.
protected override DbConnection	DbConnection [page 179]	Gets or sets the System.Data.Common.DbConnection used by this HanaCommand object.
protected override DbParameterCollection	DbParameterCollection [page 180]	Gets the collection of System.Data.Common.DbParameter objects.
protected override DbTransaction	DbTransaction [page 180]	Gets or sets the System.Data.Common.DbTransaction within which this HanaCommand object executes.
public override bool	DesignTimeVisible [page 181]	Gets or sets a value that indicates if the HanaCommand should be visible in a Microsoft Windows Form Designer control.
public new HanaParameterCollection	Parameters [page 181]	Specifies a collection of parameters for the current statement.
public new HanaTransaction	Transaction [page 182]	Specifies the HanaTransaction object in which the HanaCommand executes.
public override UpdateRowSource	UpdatedRowSource [page 183]	Gets or sets how command results are applied to the DataRow when used by the Update method of the HanaDataAdapter.

Remarks

Implements: IDbCommand, ICloneable

For more information, see Data access and manipulation.

4.1.4.4.1 HanaCommand constructor

Initializes a HanaCommand object.

Overload list

Modifier and Type	Overload name	Description
public	HanaCommand() [page 150]	Initializes a HanaCommand object.
public	HanaCommand(string) [page 150]	Initializes a HanaCommand object with a SQL string.
public	HanaCommand(string, HanaConnection) [page 151]	Initializes a HanaCommand object with a SQL string for the specified connection.
public	HanaCommand(string, HanaConnection, HanaTransaction) [page 151]	Initializes a HanaCommand object with a SQL string for the specified connection and transaction.

4.1.4.4.1.1 HanaCommand() constructor

Initializes a HanaCommand object.

Syntax

Visual Basic

```
Public Sub HanaCommand ()
```

C#

```
public HanaCommand ()
```

4.1.4.4.1.2 HanaCommand(string) constructor

Initializes a HanaCommand object with a SQL string.

Syntax

Visual Basic

```
Public Sub HanaCommand (ByVal cmdText As String)
```

C#

```
public HanaCommand (string cmdText)
```

Parameters

cmdText

The text of the SQL statement or stored procedure. For parameterized statements, use a question mark (?) placeholder to pass parameters.

4.1.4.4.1.3 HanaCommand(string, HanaConnection) constructor

Initializes a HanaCommand object with a SQL string for the specified connection.

Syntax

Visual Basic

```
Public Sub HanaCommand (
    ByVal cmdText As String,
    ByVal connection As HanaConnection
)
```

C#

```
public HanaCommand (
    string cmdText,
    HanaConnection connection
)
```

Parameters

cmdText

The text of the SQL statement or stored procedure. For parameterized statements, use a question mark (?) placeholder to pass parameters.

connection

The HanaConnection object.

4.1.4.4.1.4 HanaCommand(string, HanaConnection, HanaTransaction) constructor

Initializes a HanaCommand object with a SQL string for the specified connection and transaction.

Syntax

Visual Basic

```
Public Sub HanaCommand (
```

```
    ByVal cmdText As String,
    ByVal connection As HanaConnection,
    ByVal transaction As HanaTransaction
)
```

C#

```
public HanaCommand (
    string cmdText,
    HanaConnection connection,
    HanaTransaction transaction
)
```

Parameters

cmdText

The text of the SQL statement or stored procedure. For parameterized statements, use a question mark (?) placeholder to pass parameters.

connection

The HanaConnection object.

transaction

The HanaTransaction object in which the HanaConnection executes.

Related Information

HanaTransaction class [page 382]

4.1.4.4.2 BeginExecuteNonQuery method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand.

Overload list

Modifier and Type	Overload name	Description
public IAsyncResult	BeginExecuteNonQuery() [page 153]	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand.

Modifier and Type	Overload name	Description
public IAsyncResult	BeginExecuteNonQuery(AsyncCallback, object) [page 154]	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, given a callback procedure and state information.

4.1.4.4.2.1 BeginExecuteNonQuery() method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand.

Syntax

Visual Basic

```
Public Function BeginExecuteNonQuery () As IAsyncResult
```

C#

```
public IAsyncResult BeginExecuteNonQuery ()
```

Returns

A System.IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteNonQuery(IAsyncResult), which returns the number of affected rows.

Exceptions

HanaException class

Any error that occurred while executing the command text.

Remarks

For asynchronous command, the order of parameters must be consistent with CommandText.

Related Information

EndExecuteNonQuery(IAsyncResult) method [page 162]

4.1.4.4.2.2 BeginExecuteNonQuery(AsyncCallback, object) method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, given a callback procedure and state information.

Syntax

Visual Basic

```
Public Function BeginExecuteNonQuery (
    ByVal callback As AsyncCallback,
    ByVal stateObject As Object
) As IAsyncResult
```

C#

```
public IAsyncResult BeginExecuteNonQuery (
    AsyncCallback callback,
    object stateObject
)
```

Parameters

callback

A System.AsyncCallback delegate that is invoked when the command's execution has completed. Pass null (Nothing in Visual Basic) to indicate that no callback is required.

stateObject

A user-defined state object that is passed to the callback procedure. Retrieve this object from within the callback procedure using the System.IAsyncResult.AsyncState property.

Returns

A System.IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteNonQuery(IAsyncResult), which returns the number of affected rows.

Exceptions

HanaException class

Any error that occurred while executing the command text.

Remarks

For asynchronous command, the order of parameters must be consistent with CommandText.

Related Information

EndExecuteNonQuery(IAsyncResult) method [page 162]

4.1.4.4.3 BeginExecuteReader method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, and retrieves one or more result sets from the SAP HANA database server.

Overload list

Modifier and Type	Overload name	Description
public IAsyncResult	BeginExecuteReader() [page 156]	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, and retrieves one or more result sets from the SAP HANA database server.
public IAsyncResult	BeginExecuteReader(AsyncCallback, object) [page 157]	Initiates the asynchronous execution of a SQL statement that is described by the HanaCommand object, and retrieves the result set, given a callback procedure and state information.
public IAsyncResult	BeginExecuteReader(AsyncCallback, object, CommandBehavior) [page 158]	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, and retrieves one or more result sets from the SAP HANA database server.
public IAsyncResult	BeginExecuteReader(CommandBehavior) [page 159]	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, and retrieves one or more result sets from the SAP HANA database server.

4.1.4.4.3.1 BeginExecuteReader() method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, and retrieves one or more result sets from the SAP HANA database server.

Syntax

Visual Basic

```
Public Function BeginExecuteReader () As IAsyncResult
```

C#

```
public IAsyncResult BeginExecuteReader ()
```

Returns

A System.IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns a HanaDataReader object that can be used to retrieve the returned rows.

Exceptions

HanaException class
: Any error that occurred while executing the command text.

Remarks

For asynchronous command, the order of parameters must be consistent with CommandText.

Related Information

EndExecuteReader(IAsyncResult) method [page 165]
HanaDataReader class [page 271]

4.1.4.4.3.2 BeginExecuteReader(AsyncCallback, object) method

Initiates the asynchronous execution of a SQL statement that is described by the HanaCommand object, and retrieves the result set, given a callback procedure and state information.

Syntax

Visual Basic

```
Public Function BeginExecuteReader (
    ByVal callback As AsyncCallback,
    ByVal stateObject As Object
) As IAsyncResult
```

C#

```
public IAsyncResult BeginExecuteReader (
    AsyncCallback callback,
    object stateObject
)
```

Parameters

callback

A System.AsyncCallback delegate that is invoked when the command's execution has completed. Pass null (Nothing in Visual Basic) to indicate that no callback is required.

stateObject

A user-defined state object that is passed to the callback procedure. Retrieve this object from within the callback procedure using the System.IAsyncResult.AsyncState property.

Returns

A System.IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns a HanaDataReader object that can be used to retrieve the returned rows.

Exceptions

HanaException class

Any error that occurred while executing the command text.

Remarks

For asynchronous command, the order of parameters must be consistent with CommandText.

Related Information

EndExecuteReader(IAsyncResult) method [page 165]
HanaDataReader class [page 271]

4.1.4.4.3.3 BeginExecuteReader(AsyncCallback, object, CommandBehavior) method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, and retrieves one or more result sets from the SAP HANA database server.

Syntax

Visual Basic

```
Public Function BeginExecuteReader (
    ByVal callback As AsyncCallback,
    ByVal stateObject As Object,
    ByVal behavior As CommandBehavior
) As IAsyncResult
```

C#

```
public IAsyncResult BeginExecuteReader (
    AsyncCallback callback,
    object stateObject,
    CommandBehavior behavior
)
```

Parameters

callback

A System.AsyncCallback delegate that is invoked when the command's execution has completed. Pass null (Nothing in Visual Basic) to indicate that no callback is required.

stateObject

A user-defined state object that is passed to the callback procedure. Retrieve this object from within the callback procedure using the System.IAsyncResult.AsyncState property.

behavior

A bitwise combination of System.Data.CommandBehavior flags describing the results of the query and its effect on the connection.

Returns

A System.IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns a HanaDataReader object that can be used to retrieve the returned rows.

Exceptions

HanaException class
: Any error that occurred while executing the command text.

Remarks

For asynchronous command, the order of parameters must be consistent with CommandText.

Related Information

EndExecuteReader(IAsyncResult) method [page 165]
HanaDataReader class [page 271]

4.1.4.4.3.4 BeginExecuteReader(CommandBehavior) method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this HanaCommand, and retrieves one or more result sets from the SAP HANA database server.

Syntax

Visual Basic

```
Public Function BeginExecuteReader (ByVal behavior As
CommandBehavior) As IAsyncResult
```

C#

```
public IAsyncResult BeginExecuteReader (CommandBehavior behavior)
```

Parameters

behavior

A bitwise combination of System.Data.CommandBehavior flags describing the results of the query and its effect on the connection.

Returns

A System.IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns a HanaDataReader object that can be used to retrieve the returned rows.

Exceptions

HanaException class
: Any error that occurred while executing the command text.

Remarks

For asynchronous command, the order of parameters must be consistent with CommandText.

Related Information

EndExecuteReader(IAsyncResult) method [page 165]
HanaDataReader class [page 271]

4.1.4.4.4 Cancel() method

Cancels the execution of a HanaCommand object.

Syntax

Visual Basic
```
Public Overrides Sub Cancel ()
```

C#
```
public override void Cancel ()
```

Remarks

If there is nothing to cancel, then nothing happens. If there is a command in process, then a "Statement interrupted by user" exception is thrown.

4.1.4.4.5 CreateDbParameter() method

Creates a new instance of a System.Data.Common.DbParameter object.

Syntax

Visual Basic

```
Protected Overrides Function CreateDbParameter () As DbParameter
```

C#

```
protected override DbParameter CreateDbParameter ()
```

Returns

A System.Data.Common.DbParameter object.

4.1.4.4.6 CreateParameter() method

Provides a HanaParameter object for supplying parameters to HanaCommand objects.

Syntax

Visual Basic

```
Public Shadows Function CreateParameter () As HanaParameter
```

C#

```
public new HanaParameter CreateParameter ()
```

Returns

A new parameter, as a HanaParameter object.

Remarks

Stored procedures and some other SQL statements can take parameters, indicated in the text of a statement by a question mark (?).

The CreateParameter method provides a HanaParameter object. Set properties on the HanaParameter to specify the value, data type, and so on for the parameter.

Related Information

HanaParameter class [page 327]

4.1.4.4.7 Dispose(bool) method

Frees the resources associated with the object.

Syntax

Visual Basic

```
Protected Overrides Sub Dispose (ByVal disposing As Boolean)
```

C#

```
protected override void Dispose (bool disposing)
```

4.1.4.4.8 EndExecuteNonQuery(IAsyncResult) method

Finishes asynchronous execution of a SQL statement or stored procedure.

Syntax

Visual Basic

```
Public Function EndExecuteNonQuery (ByVal asyncResult As
IAsyncResult) As Integer
```

C#

```
public unsafe int EndExecuteNonQuery (IAsyncResult asyncResult)
```

Parameters

asyncResult

The IAsyncResult returned by the call to HanaCommand.BeginExecuteNonQuery.

Returns

The number of rows affected (the same behavior as HanaCommand.ExecuteNonQuery).

Exceptions

ArgumentException

The asyncResult parameter is null (Nothing in Visual Basic).

InvalidOperationException

The HanaCommand.EndExecuteNonQuery(IAsyncResult) was called more than once for a single command execution, or the method was mismatched against its execution method.

Remarks

Call EndExecuteNonQuery once for every call to BeginExecuteNonQuery. The call must be after BeginExecuteNonQuery has returned. ADO.NET is not thread safe; it is your responsibility to ensure that BeginExecuteNonQuery has returned. The IAsyncResult passed to EndExecuteNonQuery must be the same as the one returned from the BeginExecuteNonQuery call that is being completed. It is an error to call EndExecuteNonQuery to end a call to BeginExecuteReader, and vice versa.

If an error occurs while executing the command, then the exception is thrown when EndExecuteNonQuery is called.

There are four ways to wait for execution to complete:

(1) Call EndExecuteNonQuery.

Calling EndExecuteNonQuery blocks until the command completes. For example:

```
HanaConnection conn = new HanaConnection(
   "Server=hana-db:30015;UserID=JSmith;Password=secret");
conn.Open();
HanaCommand cmd = new HanaCommand(
   "UPDATE Departments"
       + " SET DepartmentName = 'Engineering'"
       + " WHERE DepartmentID=100",
   conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
// this will block until the command completes
int rowCount = cmd.EndExecuteNonQuery( res );
```

(2) Poll the IsCompleted property of the IAsyncResult.

For example:

```
HanaConnection conn = new HanaConnection(
   "Server=hana-db:30015;UserID=JSmith;Password=secret");
conn.Open();
HanaCommand cmd = new HanaCommand(
   "UPDATE Departments"
       + " SET DepartmentName = 'Engineering'"
       + " WHERE DepartmentID=100",
   conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
while( !res.IsCompleted ) {
   // do other work
}
// this will not block because the command is finished
int rowCount = cmd.EndExecuteNonQuery( res );
```

(3) Use the IAsyncResult.AsyncWaitHandle property to get a synchronization object, and wait on that.

For example:

```
HanaConnection conn = new HanaConnection(
   "Server=hana-db:30015;UserID=JSmith;Password=secret");
conn.Open();
HanaCommand cmd = new HanaCommand(
   "UPDATE Departments"
       + " SET DepartmentName = 'Engineering'"
       + " WHERE DepartmentID=100",
   conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// this will not block because the command is finished
int rowCount = cmd.EndExecuteNonQuery( res );
```

(4) Specify a callback function when calling BeginExecuteNonQuery.

For example:

```
private void callbackFunction( IAsyncResult ar ) {
   HanaCommand cmd = (HanaCommand) ar.AsyncState;
   // this won't block since the command has completed
   int rowCount = cmd.EndExecuteNonQuery( ar );
}
// elsewhere in the code
private void DoStuff() {
   HanaConnection conn = new HanaConnection(
    "Server=hana-db:30015;UserID=JSmith;Password=secret");
   conn.Open();
   HanaCommand cmd = new HanaCommand(
       "UPDATE Departments"
           + " SET DepartmentName = 'Engineering'"
           + " WHERE DepartmentID=100",
       conn );
   IAsyncResult res = cmd.BeginExecuteNonQuery( callbackFunction, cmd );
   // perform other work.  The callback function is
   // called when the command completes
}
```

The callback function executes in a separate thread, so the usual caveats related to updating the user interface in a threaded program apply.

Related Information

BeginExecuteNonQuery() method [page 153]

4.1.4.4.9 EndExecuteReader(IAsyncResult) method

Finishes asynchronous execution of a SQL statement or stored procedure, returning the requested HanaDataReader.

Syntax

Visual Basic

```
Public Function EndExecuteReader (ByVal asyncResult As
IAsyncResult) As HanaDataReader
```

C#

```
public unsafe HanaDataReader EndExecuteReader (IAsyncResult
asyncResult)
```

Parameters

asyncResult

The IAsyncResult returned by the call to HanaCommand.BeginExecuteReader.

Returns

A HanaDataReader object that can be used to retrieve the requested rows (the same behavior as HanaCommand.ExecuteReader).

Exceptions

ArgumentException

The asyncResult parameter is null (Nothing in Visual Basic)

InvalidOperationException

The HanaCommand.EndExecuteReader(IAsyncResult) was called more than once for a single command execution, or the method was mismatched against its execution method.

Remarks

Call EndExecuteReader once for every call to BeginExecuteReader. The call must be after BeginExecuteReader has returned. ADO.NET is not thread safe; it is your responsibility to ensure that BeginExecuteReader has returned. The IAsyncResult passed to EndExecuteReader must be the same as the one returned from the BeginExecuteReader call that is being completed. It is an error to call EndExecuteReader to end a call to BeginExecuteNonQuery, and vice versa.

If an error occurs while executing the command, then the exception is thrown when EndExecuteReader is called.

There are four ways to wait for execution to complete:

(1) Call EndExecuteReader.

Calling EndExecuteReader blocks until the command completes. For example:

```
HanaConnection conn = new HanaConnection(
   "Server=hana-db:30015;UserID=JSmith;Password=secret");
conn.Open();
HanaCommand cmd = new HanaCommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
// this blocks until the command completes
HanaDataReader reader = cmd.EndExecuteReader( res );
```

(2) Poll the IsCompleted property of the IAsyncResult.

For example:

```
HanaConnection conn = new HanaConnection(
   "Server=hana-db:30015;UserID=JSmith;Password=secret");
conn.Open();
HanaCommand cmd = new HanaCommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
while( !res.IsCompleted ) {
   // do other work
}
// this does not block because the command is finished
HanaDataReader reader = cmd.EndExecuteReader( res );
```

(3) Use the IAsyncResult.AsyncWaitHandle property to get a synchronization object.

Use the IAsyncResult.AsyncWaitHandle property to get a synchronization object, and wait on that. For example:

```
HanaConnection conn = new HanaConnection(
   "Server=hana-db:30015;UserID=JSmith;Password=secret");
conn.Open();
HanaCommand cmd = new HanaCommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// this does not block because the command is finished
HanaDataReader reader = cmd.EndExecuteReader( res );
```

(4) Specify a callback function when calling BeginExecuteReader

For example:

```
private void callbackFunction( IAsyncResult ar ) {
   HanaCommand cmd = (HanaCommand) ar.AsyncState;
   // this does not block since the command has completed
   HanaDataReader reader = cmd.EndExecuteReader();
}
// elsewhere in the code
private void DoStuff() {
   HanaConnection conn = new HanaConnection(
       "Server=hana-db:30015;UserID=JSmith;Password=secret");
   conn.Open();
   HanaCommand cmd = new HanaCommand( "SELECT * FROM Departments", conn );
   IAsyncResult res = cmd.BeginExecuteReader( callbackFunction, cmd );
   // perform other work.  The callback function is
   // called when the command completes
}
```

The callback function executes in a separate thread, so the usual caveats related to updating the user interface in a threaded program apply.

Related Information

BeginExecuteReader() method [page 156]
HanaDataReader class [page 271]

4.1.4.4.10 ExecuteDbDataReader(CommandBehavior) method

Executes the command text against the connection.

Syntax

Visual Basic

```
Protected Overrides Function ExecuteDbDataReader (ByVal behavior
As CommandBehavior) As DbDataReader
```

C#

```
protected override DbDataReader ExecuteDbDataReader
(CommandBehavior behavior)
```

Parameters

behavior

An instance of System.Data.CommandBehavior.

Returns

A System.Data.Common.DbDataReader.

4.1.4.4.11 ExecuteNonQuery() method

Executes a statement that does not return a result set, such as an INSERT, UPDATE, DELETE, or data definition statement.

Syntax

Visual Basic

```
Public Overrides Function ExecuteNonQuery () As Integer
```

C#

```
public override unsafe int ExecuteNonQuery ()
```

Returns

The number of rows affected.

Remarks

Use ExecuteNonQuery to change the data in a database without using a DataSet. Do this by executing UPDATE, INSERT, or DELETE statements.

Although ExecuteNonQuery does not return any rows, output parameters or return values that are mapped to parameters are populated with data.

For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command. For all other types of statements, and for rollbacks, the return value is -1.

Related Information

ExecuteReader() method [page 169]

4.1.4.4.12 ExecuteReader method

Executes a SQL statement that returns a result set.

Overload list

Modifier and Type	Overload name	Description
public new HanaDataReader	ExecuteReader() [page 169]	Executes a SQL statement that returns a result set.
public new HanaDataReader	ExecuteReader(CommandBehavior) [page 170]	Executes a SQL statement that returns a result set.

4.1.4.4.12.1 ExecuteReader() method

Executes a SQL statement that returns a result set.

Syntax

Visual Basic

```
Public Shadows Function ExecuteReader () As HanaDataReader
```

C#

```
public new HanaDataReader ExecuteReader ()
```

Returns

The result set as a HanaDataReader object.

Remarks

The statement is the current HanaCommand object, with CommandText and Parameters as needed. The HanaDataReader object is a read-only, forward-only result set. For modifiable result sets, use a HanaDataAdapter.

Related Information

ExecuteNonQuery() method [page 168]
HanaDataReader class [page 271]
HanaDataAdapter class [page 249]
CommandText property [page 177]
Parameters property [page 181]

4.1.4.4.12.2 ExecuteReader(CommandBehavior) method

Executes a SQL statement that returns a result set.

Syntax

Visual Basic

```
Public Shadows Function ExecuteReader (ByVal behavior As
CommandBehavior) As HanaDataReader
```

C#

```
public new HanaDataReader ExecuteReader (CommandBehavior behavior)
```

Parameters

behavior

One of CloseConnection, Default, KeyInfo, SchemaOnly, SequentialAccess, SingleResult, or SingleRow. For more information about this parameter, see the Microsoft .NET Framework documentation for CommandBehavior Enumeration.

Returns

The result set as a HanaDataReader object.

Remarks

The statement is the current HanaCommand object, with CommandText and Parameters as needed. The HanaDataReader object is a read-only, forward-only result set. For modifiable result sets, use a HanaDataAdapter.

Related Information

ExecuteNonQuery() method [page 168]
HanaDataReader class [page 271]
HanaDataAdapter class [page 249]
CommandText property [page 177]
Parameters property [page 181]

4.1.4.4.13 ExecuteReaderAsync method

An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.

Overload list

Modifier and Type	Overload name	Description
public new Task< HanaDataReader >	ExecuteReaderAsync() [page 171]	An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.
public new Task< HanaDataReader >	ExecuteReaderAsync(CancellationToken) [page 172]	An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.
public new Task< HanaDataReader >	ExecuteReaderAsync(CommandBehavior) [page 173]	An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.
public new Task< HanaDataReader >	ExecuteReaderAsync(CommandBehavior, CancellationToken) [page 174]	An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.

4.1.4.4.13.1 ExecuteReaderAsync() method

An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.

Syntax

Visual Basic

```
Public Shadows Function ExecuteReaderAsync () As Task<
HanaDataReader >
```

C#

```
public new Task< HanaDataReader > ExecuteReaderAsync ()
```

Returns

A task representing the asynchronous operation.

Exceptions

HanaException class

SAP HANA returned an error while executing the command text.

Remarks

Exceptions are reported via the returned Task object.

4.1.4.4.13.2 ExecuteReaderAsync(CancellationToken) method

An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.

Syntax

Visual Basic

```
Public Shadows Function ExecuteReaderAsync (ByVal
cancellationToken As CancellationToken) As Task< HanaDataReader >
```

C#

```
public new Task< HanaDataReader > ExecuteReaderAsync
(CancellationToken cancellationToken)
```

Parameters

cancellationToken

The cancellation instruction.

Returns

A task representing the asynchronous operation.

Exceptions

HanaException class

SAP HANA returned an error while executing the command text.

Remarks

The cancellation token can be used to request that the operation be abandoned before the command timeout elapses. Exceptions are reported via the returned Task object.

4.1.4.4.13.3 ExecuteReaderAsync(CommandBehavior) method

An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.

Syntax

Visual Basic

```
Public Shadows Function ExecuteReaderAsync (ByVal behavior As
CommandBehavior) As Task< HanaDataReader >
```

C#

```
public new Task< HanaDataReader > ExecuteReaderAsync
(CommandBehavior behavior)
```

Parameters

behavior

Options for statement execution and data retrieval.

Returns

A task representing the asynchronous operation.

Exceptions

HanaException class

SAP HANA returned an error while executing the command text.

Remarks

Exceptions are reported via the returned Task object.

4.1.4.4.13.4 ExecuteReaderAsync(CommandBehavior, CancellationToken) method

An asynchronous version of HanaCommand.ExecuteReader, which Executes a SQL statement that returns a result set.

Syntax

Visual Basic

```
Public Shadows Function ExecuteReaderAsync (
    ByVal behavior As CommandBehavior,
    ByVal cancellationToken As CancellationToken
) As Task< HanaDataReader >
```

C#

```
public new Task< HanaDataReader > ExecuteReaderAsync (
    CommandBehavior behavior,
    CancellationToken cancellationToken
)
```

Parameters

behavior

Options for statement execution and data retrieval.

cancellationToken

The cancellation instruction.

Returns

A task representing the asynchronous operation.

Exceptions

HanaException class
: SAP HANA returned an error while executing the command text.

Remarks

The cancellation token can be used to request that the operation be abandoned before the command timeout elapses. Exceptions are reported via the returned Task object.

4.1.4.4.14 ExecuteScalar() method

Executes a statement that returns a single value.

Syntax

Visual Basic
```
Public Overrides Function ExecuteScalar () As Object
```

C#
```
public override object ExecuteScalar ()
```

Returns

The first column of the first row in the result set, or a null reference if the result set is empty.

Remarks

If this method is called on a query that returns multiple rows and columns, then only the first column of the first row is returned.

4.1.4.4.15 Prepare() method

Prepares or compiles the HanaCommand on the data source.

Syntax

Visual Basic

```
Public Overrides Sub Prepare ()
```

C#

```
public override void Prepare ()
```

Remarks

If you call one of the ExecuteNonQuery, ExecuteReader, or ExecuteScalar methods after calling Prepare, then any parameter value that is larger than the value specified by the Size property is automatically truncated to the original specified size of the parameter, and no truncation errors are returned.

The truncation only happens for the following data types:

- VARCHAR
- NVARCHAR
- SHORTTEXT
- VARBINARY
- BLOB
- CLOB
- NCLOB
- TEXT

If the Size property is not specified, then the default value is used and the data is not truncated.

Related Information

ExecuteNonQuery() method [page 168]
ExecuteReader() method [page 169]
ExecuteScalar() method [page 175]

4.1.4.4.16 ResetCommandTimeout() method

Resets the CommandTimeout property to its default value of 30 seconds.

Syntax

Visual Basic

```
Public Sub ResetCommandTimeout ()
```

C#

```
public void ResetCommandTimeout ()
```

4.1.4.4.17 CommandText property

Gets or sets the text of a SQL statement or stored procedure.

Syntax

Visual Basic

```
Public Overrides Property CommandText As String
```

C#

```
public override string CommandText {get;set;}
```

Remarks

The SQL statement or the name of the stored procedure to execute. The default is an empty string.

Related Information

HanaCommand() constructor [page 150]

4.1.4.4.18 CommandTimeout property

Gets or sets a server-side timeout that cancels any statement executions that exceed a specified number of seconds in duration.

Syntax

Visual Basic

```
Public Overrides Property CommandTimeout As Integer
```

C#

```
public override int CommandTimeout {get;set;}
```

Remarks

If the statement execution duration exceeds the specified limit, then the execution call returns a HanaException back to the calling program saying that the execution has been aborted by the timeout.

To use this property, you must be connected to an SAP HANA 2.0 server.

4.1.4.4.19 CommandType property

Gets or sets the type of command represented by a HanaCommand.

Syntax

Visual Basic

```
Public Overrides Property CommandType As CommandType
```

C#

```
public override CommandType CommandType {get;set;}
```

Remarks

One of the System.Data.CommandType values. The default is System.Data.CommandType.Text.

Supported command types are as follows:

- System.Data.CommandType.StoredProcedure When you specify this CommandType, the command text must be the name of a stored procedure and you must supply any arguments as HanaParameter objects.
- System.Data.CommandType.Text This is the default value.

When the CommandType property is set to StoredProcedure, the CommandText property should be set to the name of the stored procedure. The command executes this stored procedure when you call one of the Execute methods.

Use a question mark (?) placeholder to pass parameters. For example:

```
SELECT * FROM Customers WHERE ID = ?
```

The order in which HanaParameter objects are added to the HanaParameterCollection must directly correspond to the position of the question mark placeholder for the parameter.

4.1.4.4.20 Connection property

Gets or sets the connection object to which the HanaCommand object applies.

Syntax

Visual Basic

```
Public Shadows Property Connection As HanaConnection
```

C#

```
public new HanaConnection Connection {get;set;}
```

Remarks

The default value is a null reference. In Visual Basic it is Nothing.

4.1.4.4.21 DbConnection property

Gets or sets the System.Data.Common.DbConnection used by this HanaCommand object.

Syntax

Visual Basic

```
Protected Overrides Property DbConnection As DbConnection
```

C#

```
protected override DbConnection DbConnection {get;set;}
```

Returns

The connection to the data source.

Related Information

HanaCommand class [page 147]

4.1.4.4.22 DbParameterCollection property

Gets the collection of System.Data.Common.DbParameter objects.

Syntax

Visual Basic

```
Protected ReadOnly Overrides Property DbParameterCollection As
DbParameterCollection
```

C#

```
protected override DbParameterCollection DbParameterCollection
{get;}
```

Returns

The parameters of the SQL statement or stored procedure.

4.1.4.4.23 DbTransaction property

Gets or sets the System.Data.Common.DbTransaction within which this HanaCommand object executes.

Syntax

Visual Basic

```
Protected Overrides Property DbTransaction As DbTransaction
```

C#

```
protected override DbTransaction DbTransaction {get;set;}
```

Returns

The transaction within which a Command object of a data provider executes. The default value is a null reference (Nothing in Visual Basic).

4.1.4.4.24 DesignTimeVisible property

Gets or sets a value that indicates if the HanaCommand should be visible in a Microsoft Windows Form Designer control.

Syntax

Visual Basic

```
Public Overrides Property DesignTimeVisible As Boolean
```

C#

```
public override bool DesignTimeVisible {get;set;}
```

Remarks

The default is true.

True if this HanaCommand instance should be visible, false if this instance should not be visible. The default is false.

4.1.4.4.25 Parameters property

Specifies a collection of parameters for the current statement.

Syntax

Visual Basic

```
Public ReadOnly Shadows Property Parameters As
HanaParameterCollection
```

C#

```
public new HanaParameterCollection Parameters {get;}
```

Remarks

Use question marks in the CommandText to indicate parameters.

The parameters of the SQL statement or stored procedure. The default value is an empty collection.

When CommandType is set to Text, pass parameters using the question mark placeholder. For example:

```
SELECT * FROM Customers WHERE ID = ?
```

The order in which HanaParameter objects are added to the HanaParameterCollection must directly correspond to the position of the question mark placeholder for the parameter in the command text.

When the parameters in the collection do not match the requirements of the query to be executed, an error may result or an exception may be thrown.

Related Information

HanaParameterCollection class [page 342]

4.1.4.4.26 Transaction property

Specifies the HanaTransaction object in which the HanaCommand executes.

Syntax

Visual Basic

```
Public Shadows Property Transaction As HanaTransaction
```

C#

```
public new HanaTransaction Transaction {get;set;}
```

Remarks

The default value is a null reference. In Visual Basic, this is Nothing.

You cannot set the Transaction property if it is already set to a specific value and the command is executing. If you set the transaction property to a HanaTransaction object that is not connected to the same HanaConnection object as the HanaCommand object, then an exception is thrown the next time that you attempt to execute a statement.

For more information, see Transaction processing.

Related Information

HanaTransaction class [page 382]

4.1.4.4.27 UpdatedRowSource property

Gets or sets how command results are applied to the DataRow when used by the Update method of the HanaDataAdapter.

Syntax

Visual Basic

```
Public Overrides Property UpdatedRowSource As UpdateRowSource
```

C#

```
public override UpdateRowSource UpdatedRowSource {get;set;}
```

Remarks

One of the UpdatedRowSource values. The default value is UpdateRowSource.OutputParameters. If the command is automatically generated, then this property is UpdateRowSource.None.

UpdatedRowSource.Both, which returns both resultset and output parameters, is not supported.

4.1.4.5 HanaCommandBuilder class

A way to generate single-table SQL statements that reconcile changes made to a DataSet with the data in the associated database.

Syntax

Visual Basic

```
Public NotInheritable Class HanaCommandBuilder Inherits
System.Data.Common.DbCommandBuilder
```

C#

```
public sealed class HanaCommandBuilder :
System.Data.Common.DbCommandBuilder
```

Members

All members of HanaCommandBuilder, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaCommandBuilder [page 185]	Initializes a HanaCommandBuilder object.

Methods

Modifier and Type	Method	Description
protected override void	ApplyParameterInfo(DbParameter, DataRow, StatementType, bool) [page 186]	Allows the provider implementation of System.Data.Common.DbCommandBuilder to handle additional parameter properties.
public static void	DeriveParameters(HanaCommand) [page 187]	Populates the Parameters collection of the specified HanaCommand object.
public new HanaCommand	GetDeleteCommand [page 188]	Returns the generated HanaCommand object that performs DELETE operations on the database when HanaDataAdapter.Update is called.
public new HanaCommand	GetInsertCommand [page 190]	Returns the generated HanaCommand object that performs INSERT operations on the database when an Update is called.
protected override string	GetParameterName [page 192]	Returns the name of the specified parameter in the format of @p#.
protected override string	GetParameterPlaceholder(int) [page 194]	Returns the placeholder for the parameter in the associated SQL statement.
protected override DataTable	GetSchemaTable(DbCommand) [page 195]	Returns the schema table for the HanaCommandBuilder object.
public new HanaCommand	GetUpdateCommand [page 196]	Returns the generated HanaCommand object that performs UPDATE operations on the database when an Update is called.
protected override DbCommand	InitializeCommand(DbCommand) [page 198]	Resets the System.Data.Common.DbCommand.CommandTimeout, System.Data.Common.DbCommand.Transaction, System.Data.Common.DbCommand.CommandType, and System.Data.Common.DbCommand.UpdatedRowSource properties on the System.Data.Common.DbCommand.
public override string	QuoteIdentifier(string) [page 199]	Returns the correct quoted form of an unquoted identifier, including properly escaping any embedded quotes in the identifier.
protected override void	SetRowUpdatingHandler(DbDataAdapter) [page 199]	Registers the HanaCommandBuilder object to handle the HanaDataAdapter.RowUpdating event for a HanaDataAdapter object.

Modifier and Type	Method	Description
public override string	UnquoteIdentifier(string) [page 200]	Returns the correct unquoted form of a quoted identifier, including properly unescaping any embedded quotes in the identifier.

Properties

Modifier and Type	Property	Description
public new HanaDataAdapter	DataAdapter [page 201]	Specifies the HanaDataAdapter for which to generate statements.
public override string	QuotePrefix [page 201]	Gets or sets the beginning character or characters to use when specifying database objects whose names contain characters such as spaces.
public override string	QuoteSuffix [page 202]	Gets or sets the ending character or characters to use when specifying database objects whose names contains characters such as spaces.

4.1.4.5.1 HanaCommandBuilder constructor

Initializes a HanaCommandBuilder object.

Overload list

Modifier and Type	Overload name	Description
public	HanaCommandBuilder() [page 185]	Initializes a HanaCommandBuilder object.
public	HanaCommandBuilder(HanaDataAdapter) [page 186]	Initializes a HanaCommandBuilder object.

4.1.4.5.1.1 HanaCommandBuilder() constructor

Initializes a HanaCommandBuilder object.

Syntax

Visual Basic

```
Public Sub HanaCommandBuilder ()
```

C#

```
public HanaCommandBuilder ()
```

4.1.4.5.1.2 HanaCommandBuilder(HanaDataAdapter) constructor

Initializes a HanaCommandBuilder object.

Syntax

Visual Basic

```
Public Sub HanaCommandBuilder (ByVal adapter As HanaDataAdapter)
```

C#

```
public HanaCommandBuilder (HanaDataAdapter adapter)
```

Parameters

adapter

A HanaDataAdapter object for which to generate reconciliation statements.

4.1.4.5.2 ApplyParameterInfo(DbParameter, DataRow, StatementType, bool) method

Allows the provider implementation of System.Data.Common.DbCommandBuilder to handle additional parameter properties.

Syntax

Visual Basic

```
Protected Overrides Sub ApplyParameterInfo (
    ByVal parameter As DbParameter,
    ByVal row As DataRow,
    ByVal statementType As StatementType,
    ByVal whereClause As Boolean
)
```

C#

```
protected override void ApplyParameterInfo (
    DbParameter parameter,
    DataRow row,
    StatementType statementType,
    bool whereClause
)
```

Parameters

parameter

A System.Data.Common.DbParameter to which the additional modifications are applied.

row

The System.Data.DataRow from the schema table provided by HanaDataReader.GetSchemaTable.

statementType

The type of command being generated: INSERT, UPDATE or DELETE.

whereClause

The value is true if the parameter is part of the UPDATE or DELETE WHERE clause, and false if it is part of the INSERT or UPDATE values.

Related Information

GetSchemaTable() method [page 292]

4.1.4.5.3 DeriveParameters(HanaCommand) method

Populates the Parameters collection of the specified HanaCommand object.

Syntax

Visual Basic

```
Public Shared Sub DeriveParameters (ByVal command As HanaCommand)
```

C#

```
public static void DeriveParameters (HanaCommand command)
```

Parameters

command

A HanaCommand object for which to derive parameters.

Remarks

This method is used for the stored procedure specified in the HanaCommand.

DeriveParameters overwrites any existing parameter information for the HanaCommand.

DeriveParameters requires an extra call to the SAP HANA database server. If the parameter information is known in advance, then it is more efficient to populate the Parameters collection by setting the information explicitly.

4.1.4.5.4 GetDeleteCommand method

Returns the generated HanaCommand object that performs DELETE operations on the database when HanaDataAdapter.Update is called.

Overload list

Modifier and Type	Overload name	Description
public new HanaCommand	GetDeleteCommand() [page 188]	Returns the generated HanaCommand object that performs DELETE operations on the database when HanaDataAdapter.Update is called.
public new HanaCommand	GetDeleteCommand(bool) [page 189]	Returns the generated HanaCommand object that performs DELETE operations on the database when HanaDataAdapter.Update is called.

4.1.4.5.4.1 GetDeleteCommand() method

Returns the generated HanaCommand object that performs DELETE operations on the database when HanaDataAdapter.Update is called.

Syntax

Visual Basic

```
Public Shadows Function GetDeleteCommand () As HanaCommand
```

C#

```
public new HanaCommand GetDeleteCommand ()
```

Returns

The automatically generated HanaCommand object required to perform deletions.

Remarks

The GetDeleteCommand method returns the HanaCommand object to be executed, so it is useful for informational or troubleshooting purposes.

Alternatively, use GetDeleteCommand as the basis of a modified command. For example, you might call GetDeleteCommand and modify the CommandTimeout value, and then explicitly set that value on the HanaDataAdapter.

SQL statements are first generated when the application calls Update or GetDeleteCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetDeleteCommand continues to use information from the previous statement.

4.1.4.5.4.2 GetDeleteCommand(bool) method

Returns the generated HanaCommand object that performs DELETE operations on the database when HanaDataAdapter.Update is called.

Syntax

Visual Basic

```
Public Shadows Function GetDeleteCommand (ByVal
useColumnsForParameterNames As Boolean) As HanaCommand
```

C#

```
public new HanaCommand GetDeleteCommand (bool
useColumnsForParameterNames)
```

Parameters

useColumnsForParameterNames

If true, then generate parameter names matching column names if possible. If false, then generate @p1, @p2, and so on.

Returns

The automatically generated HanaCommand object required to perform deletions.

Remarks

The GetDeleteCommand method returns the HanaCommand object to be executed, so it is useful for informational or troubleshooting purposes.

Alternatively, use GetDeleteCommand as the basis of a modified command. For example, you might call GetDeleteCommand and modify the CommandTimeout value, and then explicitly set that value on the HanaDataAdapter.

SQL statements are first generated when the application calls Update or GetDeleteCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetDeleteCommand continues to use information from the previous statement.

4.1.4.5.5 GetInsertCommand method

Returns the generated HanaCommand object that performs INSERT operations on the database when an Update is called.

Overload list

Modifier and Type	Overload name	Description
public new HanaCommand	GetInsertCommand() [page 190]	Returns the generated HanaCommand object that performs INSERT operations on the database when an Update is called.
public new HanaCommand	GetInsertCommand(bool) [page 191]	Returns the generated HanaCommand object that performs INSERT operations on the database when an Update is called.

4.1.4.5.5.1 GetInsertCommand() method

Returns the generated HanaCommand object that performs INSERT operations on the database when an Update is called.

Syntax

Visual Basic

```
Public Shadows Function GetInsertCommand () As HanaCommand
```

C#

```
public new HanaCommand GetInsertCommand ()
```

Returns

The automatically generated HanaCommand object required to perform insertions.

Remarks

The GetInsertCommand method returns the HanaCommand object to be executed, so it is useful for informational or troubleshooting purposes.

Alternatively, use GetInsertCommand as the basis of a modified command. For example, you might call GetInsertCommand and modify the CommandTimeout value, and then explicitly set that value on the HanaDataAdapter.

SQL statements are first generated either when the application calls Update or GetInsertCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetInsertCommand continues to use information from the previous statement, which might not be correct.

Related Information

GetDeleteCommand() method [page 188]

4.1.4.5.5.2 GetInsertCommand(bool) method

Returns the generated HanaCommand object that performs INSERT operations on the database when an Update is called.

Syntax

Visual Basic

```
Public Shadows Function GetInsertCommand (ByVal
useColumnsForParameterNames As Boolean) As HanaCommand
```

C#

```
public new HanaCommand GetInsertCommand (bool
useColumnsForParameterNames)
```

Parameters

useColumnsForParameterNames

If true, then generate parameter names matching column names if possible. If false, then generate @p1, @p2, and so on.

Returns

The automatically generated HanaCommand object required to perform insertions.

Remarks

The GetInsertCommand method returns the HanaCommand object to be executed, so it is useful for informational or troubleshooting purposes.

Alternatively, use GetInsertCommand as the basis of a modified command. For example, you might call GetInsertCommand and modify the CommandTimeout value, and then explicitly set that value on the HanaDataAdapter.

SQL statements are first generated either when the application calls Update or GetInsertCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetInsertCommand continues to use information from the previous statement, which might not be correct.

Related Information

GetDeleteCommand() method [page 188]

4.1.4.5.6 GetParameterName method

Returns the name of the specified parameter in the format of @p#.

Overload list

Modifier and Type	Overload name	Description
protected override string	GetParameterName(int) [page 193]	Returns the name of the specified parameter in the format of @p#.
protected override string	GetParameterName(string) [page 193]	Returns the full parameter name, given the partial parameter name.

4.1.4.5.6.1 GetParameterName(int) method

Returns the name of the specified parameter in the format of @p#.

Syntax

Visual Basic

```
Protected Overrides Function GetParameterName (ByVal index As
Integer) As String
```

C#

```
protected override string GetParameterName (int index)
```

Parameters

index

The number to be included as part of the parameter's name.

Returns

The name of the parameter with the specified number appended as part of the parameter name.

Remarks

Use when building a custom command builder.

4.1.4.5.6.2 GetParameterName(string) method

Returns the full parameter name, given the partial parameter name.

Syntax

Visual Basic

```
Protected Overrides Function GetParameterName (ByVal parameterName
As String) As String
```

C#

```
protected override string GetParameterName (string parameterName)
```

Parameters

parameterName

The partial name of the parameter.

Returns

The full parameter name corresponding to the partial parameter name requested.

4.1.4.5.7 GetParameterPlaceholder(int) method

Returns the placeholder for the parameter in the associated SQL statement.

Syntax

Visual Basic

```
Protected Overrides Function GetParameterPlaceholder (ByVal index
As Integer) As String
```

C#

```
protected override string GetParameterPlaceholder (int index)
```

Parameters

index

The number to be included as part of the parameter's name.

Returns

The name of the parameter with the specified number appended.

4.1.4.5.8 GetSchemaTable(DbCommand) method

Acts as an override for the regular DbCommandBuilder.GetSchemaTable method in the .NET Framework and .NET Core.

Syntax

Visual Basic

```
Protected Overrides Function GetSchemaTable (ByVal sourceCommand
As DbCommand) As DataTable
```

C#

```
protected override DataTable GetSchemaTable (DbCommand
sourceCommand)
```

Parameters

sourceCommand

The System.Data.Common.DbCommand for which to retrieve the corresponding schema table.

Returns

This method takes a DbCommand and returns a DataTable with details about the schema for the passed in DBCommand.

Related Information

HanaCommandBuilder class [page 183]

4.1.4.5.9 GetUpdateCommand method

Returns the generated HanaCommand object that performs UPDATE operations on the database when an Update is called.

Overload list

Modifier and Type	Overload name	Description
public new HanaCommand	GetUpdateCommand() [page 196]	Returns the generated HanaCommand object that performs UPDATE operations on the database when an Update is called.
public new HanaCommand	GetUpdateCommand(bool) [page 197]	

4.1.4.5.9.1 GetUpdateCommand() method

Returns the generated HanaCommand object that performs UPDATE operations on the database when an Update is called.

Syntax

Visual Basic

```
Public Shadows Function GetUpdateCommand () As HanaCommand
```

C#

```
public new HanaCommand GetUpdateCommand ()
```

Returns

The automatically generated HanaCommand object required to perform updates.

Remarks

The GetUpdateCommand method returns the HanaCommand object to be executed, so it is useful for informational or troubleshooting purposes.

Alternatively, use GetUpdateCommand as the basis of a modified command. For example, you might call GetUpdateCommand and modify the CommandTimeout value, and then explicitly set that value on the HanaDataAdapter.

SQL statements are first generated when the application calls Update or GetUpdateCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetUpdateCommand continues to use information from the previous statement, which might not be correct.

4.1.4.5.9.2 GetUpdateCommand(bool) method

Returns the generated HanaCommand object that performs UPDATE operations on the database when an Update is called.

Syntax

Visual Basic

```
Public Shadows Function GetUpdateCommand (ByVal
useColumnsForParameterNames As Boolean) As HanaCommand
```

C#

```
public new HanaCommand GetUpdateCommand (bool
useColumnsForParameterNames)
```

Parameters

useColumnsForParameterNames

If true, then generate parameter names matching column names if possible. If false, then generate @p1, @p2, and so on.

Returns

The automatically generated HanaCommand object required to perform updates.

Remarks

The GetUpdateCommand method returns the HanaCommand object to be executed, so it is useful for informational or troubleshooting purposes.

Alternatively, use GetUpdateCommand as the basis of a modified command. For example, you might call GetUpdateCommand and modify the CommandTimeout value, and then explicitly set that value on the HanaDataAdapter.

SQL statements are first generated when the application calls Update or GetUpdateCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement

in any way. Otherwise, the GetUpdateCommand continues to use information from the previous statement, which might not be correct.

4.1.4.5.10 InitializeCommand(DbCommand) method

Resets the System.Data.Common.DbCommand.CommandTimeout, System.Data.Common.DbCommand.Transaction, System.Data.Common.DbCommand.CommandType, and System.Data.Common.DbCommand.UpdatedRowSource properties on the System.Data.Common.DbCommand.

Syntax

Visual Basic

```
Protected Overrides Function InitializeCommand (ByVal command As
DbCommand) As DbCommand
```

C#

```
protected override DbCommand InitializeCommand (DbCommand command)
```

Parameters

command

The System.Data.Common.DbCommand to be used by the command builder for the corresponding insert, update, or delete command.

Returns

A System.Data.Common.DbCommand instance to use for each insert, update, or delete operation. Passing a null value allows the InitializeCommand method to create a System.Data.Common.DbCommand object based on the SELECT statement associated with the HanaCommandBuilder object.

Related Information

HanaCommandBuilder class [page 183]

4.1.4.5.11 QuoteIdentifier(string) method

Returns the correct quoted form of an unquoted identifier, including properly escaping any embedded quotes in the identifier.

Syntax

Visual Basic

```
Public Overrides Function QuoteIdentifier (ByVal
unquotedIdentifier As String) As String
```

C#

```
public override string QuoteIdentifier (string unquotedIdentifier)
```

Parameters

unquotedIdentifier

The string representing the unquoted identifier that will have to be quoted.

Returns

Returns a string representing the quoted form of an unquoted identifier with embedded quotes properly escaped.

4.1.4.5.12 SetRowUpdatingHandler(DbDataAdapter) method

Registers the HanaCommandBuilder object to handle the HanaDataAdapter.RowUpdating event for a HanaDataAdapter object.

Syntax

Visual Basic

```
Protected Overrides Sub SetRowUpdatingHandler (ByVal adapter As
DbDataAdapter)
```

C#

```
protected override void SetRowUpdatingHandler (DbDataAdapter
adapter)
```

Parameters

adapter

The HanaDataAdapter object to be used for the update.

Related Information

HanaCommandBuilder class [page 183]
HanaDataAdapter class [page 249]
RowUpdating event [page 271]

4.1.4.5.13 UnquoteIdentifier(string) method

Returns the correct unquoted form of a quoted identifier, including properly un-escaping any embedded quotes in the identifier.

Syntax

Visual Basic

```
Public Overrides Function UnquoteIdentifier (ByVal
quotedIdentifier As String) As String
```

C#

```
public override string UnquoteIdentifier (string quotedIdentifier)
```

Parameters

quotedIdentifier

The string representing the quoted identifier that will have its embedded quotes removed.

Returns

Returns a string representing the unquoted form of a quoted identifier with embedded quotes properly un-escaped.

4.1.4.5.14 DataAdapter property

Specifies the HanaDataAdapter for which to generate statements.

Syntax

Visual Basic

```
Public Shadows Property DataAdapter As HanaDataAdapter
```

C#

```
public new HanaDataAdapter DataAdapter {get;set;}
```

Remarks

A HanaDataAdapter object.

When you create a new instance of HanaCommandBuilder, any existing HanaCommandBuilder that is associated with this HanaDataAdapter is released.

4.1.4.5.15 QuotePrefix property

Gets or sets the beginning character or characters to use when specifying database objects whose names contain characters such as spaces.

Syntax

Visual Basic

```
Public Overrides Property QuotePrefix As String
```

C#

```
public override string QuotePrefix {get;set;}
```

Remarks

Some database objects contain characters such as spaces, commas, and semicolons. Use the QuotePrefix and QuoteSuffix properties to specify delimiters such as double quotes, left brackets, and right brackets to enclose the object name.

The default value is an empty string.

If you set this property to double quotes (`""`), then the HanaCommandBuild class double quotes table names and column names.

Exceptions

InvalidOperationException	This property cannot be changed after an insert, update, or delete command has been generated.

Example

The following example sets both Quote Prefix and QuoteSuffix to use double quotes. The HanaCommandBuilder class double quotes all table names and columns names for generated InsertCommands, UpdateCommands, and DeleteCommands.

```
var cmb = new HanaCommandBuilder(adapter);
cmb.QuotePrefix = "\"";
cmb.QuoteSuffix = "\"";
```

4.1.4.5.16 QuoteSuffix property

Gets or sets the ending character or characters to use when specifying database objects whose names contain characters such as spaces.

Syntax

Visual Basic

```
Public Overrides Property QuoteSuffix As String
```

C#

```
public override string QuoteSuffix {get;set;}
```

Remarks

Some database objects contain characters such as spaces, commas, and semicolons. Use the QuotePrefix and QuoteSuffix properties to specify delimiters such as double quotes, left brackets, and right brackets to enclose the object name.

The default value is an empty string.

If you set this property to double quotes (`""`), then the HanaCommandBuild class double quotes table names and column names.

Exceptions

InvalidOperationException	This property cannot be changed after an insert, update, or delete command has been generated.

Example

The following example sets both Quote Prefix and QuoteSuffix to use double quotes. The HanaCommandBuilder class double quotes all table names and columns names for generated InsertCommands, UpdateCommands, and DeleteCommands.

```
var cmb = new HanaCommandBuilder(adapter);
cmb.QuotePrefix = "\"";
cmb.QuoteSuffix = "\"";
```

4.1.4.6 HanaConnection class

Represents a connection to an SAP HANA database.

Syntax

Visual Basic

```
Public NotInheritable Class HanaConnection Inherits
System.Data.Common.DbConnection
```

C#

```
public sealed class HanaConnection :
System.Data.Common.DbConnection
```

Members

All members of HanaConnection, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaConnection [page 205]	Initializes a HanaConnection object.

Methods

Modifier and Type	Method	Description
protected override DbTransaction	BeginDbTransaction(IsolationLevel) [page 208]	Starts a database transaction.
public new HanaTransaction	BeginTransaction [page 208]	Returns a transaction object.
public override void	ChangeDatabase(string) [page 212]	Changes the current database for an open HanaConnection.
public static void	ChangePassword(string, string) [page 212]	Changes the password to the supplied new password for the user indicated in the connection string.
public static void	ClearAllPools() [page 213]	Empties all connection pools.
public static void	ClearPool(HanaConnection) [page 213]	Empties the connection pool associated with the specified connection.
public override void	Close() [page 214]	Closes a database connection.
public new HanaCommand	CreateCommand() [page 214]	Initializes a HanaCommand object.
protected override DbCommand	CreateDbCommand() [page 215]	Creates and returns a System.Data.Common.DbCommand object associated with the current connection.
protected override void	Dispose(bool) [page 215]	Frees the resources associated with the object.
public void	EnlistDistributedTransaction(System.EnterpriseServices.ITransaction) [page 216]	Enlists in the specified transaction as a distributed transaction.
public override void	EnlistTransaction(System.Transactions.Transaction) [page 216]	Enlists in the specified transaction as a distributed transaction.
public override DataTable	GetSchema [page 217]	Returns the list of supported schema collections.
public override unsafe void	Open() [page 224]	Opens a database connection with the property settings specified by the HanaConnection.ConnectionString.

Properties

Modifier and Type	Property	Description
public override string	ConnectionString [page 224]	Provides the database connection string.
public override int	ConnectionTimeout [page 226]	Gets the number of seconds before a connection attempt times out with an error.
public HanaCredential	Credential [page 226]	Gets or sets the HanaCredential object for this connection.
public override string	Database [page 227]	Gets the SAP HANA database server name (for example, GA1).

Modifier and Type	Property	Description
public override string	DataSource [page 227]	Gets the SAP HANA database server host name (for example, hana-db).
public string	InitString [page 228]	A SQL statement that is executed immediately after the connection is established.
public override string	ServerVersion [page 228]	Gets a string that contains the version of the instance of SAP HANA to which the client is connected.
public override ConnectionState	State [page 229]	Indicates the state of the HanaConnection object.

Events

Modifier and Type	Event	Description
public HanaInfoMessageEventHandler	InfoMessage [page 229]	Occurs when the SAP HANA database server returns a warning or informational message.
public override StateChangeEventHandler	StateChange [page 230]	Occurs when the state of the HanaConnection object changes.

Remarks

For a list of connection parameters, see Connection parameters.

4.1.4.6.1 HanaConnection constructor

Initializes a HanaConnection object.

Overload list

Modifier and Type	Overload name	Description
public	HanaConnection() [page 206]	Initializes a HanaConnection object.
public	HanaConnection(string) [page 206]	Initializes a HanaConnection object.
public	HanaConnection(string, HanaCredential) [page 207]	Initializes a new instance of the HanaConnection class given a connection string, and a HanaCredential object that contains the user ID and password.

4.1.4.6.1.1 HanaConnection() constructor

Initializes a HanaConnection object.

Syntax

Visual Basic

```
Public Sub HanaConnection ()
```

C#

```
public HanaConnection ()
```

Remarks

The connection must be opened before you can perform any operations against the database.

4.1.4.6.1.2 HanaConnection(string) constructor

Initializes a HanaConnection object.

Syntax

Visual Basic

```
Public Sub HanaConnection (ByVal connectionString As String)
```

C#

```
public HanaConnection (string connectionString)
```

Parameters

connectionString

A SAP HANA connection string. A connection string is a semicolon-separated list of keyword=value pairs. For a list of connection parameters, see Connection parameters.

Remarks

The connection must then be opened before you can perform any operations against the database.

Example

The following statement initializes a HanaConnection object for a connection to an SAP HANA database server named HumanResources. The connection uses the user ID Admin and the password Money123.

```
HanaConnection conn = new HanaConnection(
   "userid=Admin;password=Money123;server=HumanResources:30015" );
conn.Open();
```

Related Information

HanaConnection class [page 203]

4.1.4.6.1.3 HanaConnection(string, HanaCredential) constructor

Initializes a new instance of the HanaConnection class given a connection string, and a HanaCredential object that contains the user ID and password. This constructor is not supported for .NET Core.

Syntax

Visual Basic

```
Public Sub HanaConnection (
    ByVal connectionString As String,
    ByVal credential As HanaCredential
)
```

C#

```
public HanaConnection (
    string connectionString,
    HanaCredential credential
)
```

Parameters

connectionString

An SAP HANA connection string. A connection string is a semicolon-separated list of keyword=value pairs. For a list of connection parameters, see Connection parameters.

credential

A HanaCredential object. If credential is null, then HanaConnection(String, HanaCredential) is functionally equivalent to HanaConnection(String).

Related Information

HanaConnection class [page 203]

4.1.4.6.2 BeginDbTransaction(IsolationLevel) method

Starts a database transaction.

Syntax

Visual Basic

```
Protected Overrides Function BeginDbTransaction (ByVal
isolationLevel As IsolationLevel) As DbTransaction
```

C#

```
protected override DbTransaction BeginDbTransaction
(IsolationLevel isolationLevel)
```

Parameters

isolationLevel

Specifies the isolation level for the transaction.

Returns

An object representing the new transaction.

4.1.4.6.3 BeginTransaction method

Returns a transaction object.

Overload list

Modifier and Type	Overload name	Description
public new HanaTransaction	BeginTransaction() [page 209]	Returns a transaction object.

Modifier and Type	Overload name	Description
public HanaTransaction	BeginTransaction(HanaIsolationLevel) [page 210]	Returns a transaction object.
public new HanaTransaction	BeginTransaction(IsolationLevel) [page 211]	Returns a transaction object.

4.1.4.6.3.1 BeginTransaction() method

Returns a transaction object.

Syntax

Visual Basic

```
Public Shadows Function BeginTransaction () As HanaTransaction
```

C#

```
public new HanaTransaction BeginTransaction ()
```

Returns

A HanaTransaction object representing the new transaction.

Remarks

Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a call to the Commit or Rollback methods.

To associate a command with a transaction object, use the HanaCommand.Transaction property.

Related Information

HanaTransaction class [page 382]
Transaction property [page 182]

4.1.4.6.3.2 BeginTransaction(HanaIsolationLevel) method

Returns a transaction object.

Syntax

Visual Basic

```
Public Function BeginTransaction (ByVal isolationLevel As
HanaIsolationLevel) As HanaTransaction
```

C#

```
public HanaTransaction BeginTransaction (HanaIsolationLevel
isolationLevel)
```

Parameters

isolationLevel

A member of the HanaIsolationLevel enumeration. The default value is ReadCommitted.

Returns

A HanaTransaction object representing the new transaction.

Remarks

Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a call to the Commit or Rollback methods.

To associate a command with a transaction object, use the HanaCommand.Transaction property.

For more information, see Transaction processing.

Related Information

HanaTransaction class [page 382]
Transaction property [page 182]
HanaIsolationLevel enumeration [page 392]
Commit() method [page 384]
Rollback() method [page 384]

4.1.4.6.3.3 BeginTransaction(IsolationLevel) method

Returns a transaction object.

Syntax

Visual Basic

```
Public Shadows Function BeginTransaction (ByVal isolationLevel As
IsolationLevel) As HanaTransaction
```

C#

```
public new HanaTransaction BeginTransaction (IsolationLevel
isolationLevel)
```

Parameters

isolationLevel

A member of the HanaIsolationLevel enumeration. The default value is ReadCommitted.

Returns

A HanaTransaction object representing the new transaction.

Remarks

Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a call to the Commit or Rollback methods.

To associate a command with a transaction object, use the HanaCommand.Transaction property.

Example

```
HanaTransaction tx =
 conn.BeginTransaction( HanaIsolationLevel.ReadUncommitted );
```

Related Information

HanaTransaction class [page 382]
Transaction property [page 182]
HanaIsolationLevel enumeration [page 392]

4.1.4.6.4 ChangeDatabase(string) method

Changes the current database for an open HanaConnection.

Syntax

Visual Basic

```
Public Overrides Sub ChangeDatabase (ByVal database As String)
```

C#

```
public override void ChangeDatabase (string database)
```

Parameters

database

The name of the database to use instead of the current database.

4.1.4.6.5 ChangePassword(string, string) method

Changes the password to the supplied new password for the user indicated in the connection string.

Syntax

Visual Basic

```
Public Shared Sub ChangePassword (
    ByVal connectionString As String,
    ByVal newPassword As String
)
```

C#

```
public static void ChangePassword (
    string connectionString,
    string newPassword
)
```

Parameters

connectionString

The connection string that contains enough information to connect to the SAP HANA database server that you want. The connection string may contain the user ID and the current password.

newPassword

The new password to set. This password must comply with any password security policy set on the SAP HANA database server, including minimum length, requirements for specific characters, and so on.

Exceptions

ArgumentNullException

Either the connectionString or the newPassword parameter is null.

ArgumentException

The connection string includes the option to use integrated security.

4.1.4.6.6 ClearAllPools() method

Empties all connection pools.

Syntax

Visual Basic

```
Public Shared Sub ClearAllPools ()
```

C#

```
public static void ClearAllPools ()
```

4.1.4.6.7 ClearPool(HanaConnection) method

Empties the connection pool associated with the specified connection.

Syntax

Visual Basic

```
Public Shared Sub ClearPool (ByVal connection As HanaConnection)
```

C#

```
public static void ClearPool (HanaConnection connection)
```

Parameters

connection

The HanaConnection object to be cleared from the pool.

Related Information

HanaConnection class [page 203]

4.1.4.6.8 Close() method

Closes a database connection.

Syntax

Visual Basic

```
Public Overrides Sub Close ()
```

C#

```
public override void Close ()
```

Remarks

The Close method rolls back any pending transactions. It then releases the connection to the connection pool, or closes the connection if connection pooling is disabled. If Close is called while handling a StateChange event, then no additional StateChange events are fired. An application can call Close multiple times.

4.1.4.6.9 CreateCommand() method

Initializes a HanaCommand object.

Syntax

Visual Basic

```
Public Shadows Function CreateCommand () As HanaCommand
```

C#

```
public new HanaCommand CreateCommand ()
```

Returns

A HanaCommand object.

Remarks

The command object is associated with the HanaConnection object.

Related Information

HanaCommand class [page 147]
HanaConnection class [page 203]

4.1.4.6.10 CreateDbCommand() method

Creates and returns a System.Data.Common.DbCommand object associated with the current connection.

Syntax

Visual Basic

```
Protected Overrides Function CreateDbCommand () As DbCommand
```

C#

```
protected override DbCommand CreateDbCommand ()
```

Returns

A System.Data.Common.DbCommand object.

4.1.4.6.11 Dispose(bool) method

Frees the resources associated with the object.

Syntax

Visual Basic

```
Protected Overrides Sub Dispose (ByVal disposing As Boolean)
```

C#

```
protected override void Dispose (bool disposing)
```

4.1.4.6.12 EnlistDistributedTransaction(System.EnterpriseServices.ITransaction) method

Enlists in the specified transaction as a distributed transaction.

Syntax

Visual Basic

```
Public Sub EnlistDistributedTransaction (ByVal transaction As
System.EnterpriseServices.ITransaction)
```

C#

```
public void EnlistDistributedTransaction
(System.EnterpriseServices.ITransaction transaction)
```

Parameters

transaction

A reference to an existing System.EnterpriseServices.ITransaction in which to enlist.

4.1.4.6.13 EnlistTransaction(System.Transactions.Transaction) method

Enlists in the specified transaction as a distributed transaction.

Syntax

Visual Basic

```
Public Overrides Sub EnlistTransaction (ByVal transaction As
System.Transactions.Transaction)
```

C#

```
public override void EnlistTransaction
(System.Transactions.Transaction transaction)
```

Parameters

transaction

A reference to an existing System.Transactions.Transaction in which to enlist.

4.1.4.6.14 GetSchema method

Returns the list of supported schema collections.

Overload list

Modifier and Type	Overload name	Description
public override DataTable	GetSchema() [page 217]	Returns the list of supported schema collections.
public override DataTable	GetSchema(string) [page 218]	Returns information for the specified metadata collection for this HanaConnection object.
public override DataTable	GetSchema(string, string[]) [page 218]	Returns schema information for the data source of this HanaConnection object and, if specified, uses the specified string for the schema name and the specified string array for the restriction values.

4.1.4.6.14.1 GetSchema() method

Returns the list of supported schema collections.

Syntax

Visual Basic

```
Public Overrides Function GetSchema () As DataTable
```

C#

```
public override DataTable GetSchema ()
```

Remarks

See GetSchema(string,string[]) for a description of the available metadata.

4.1.4.6.14.2 GetSchema(string) method

Returns information for the specified metadata collection for this HanaConnection object.

Syntax

Visual Basic

```
Public Overrides Function GetSchema (ByVal collection As String)
As DataTable
```

C#

```
public override DataTable GetSchema (string collection)
```

Parameters

collection

Name of the metadata collection. If a name is not provided, then MetaDataCollections is used.

Remarks

See GetSchema(string,string[]) for a description of the available metadata.

Related Information

HanaConnection class [page 203]

4.1.4.6.14.3 GetSchema(string, string[]) method

Returns schema information for the data source of this HanaConnection object and, if specified, uses the specified string for the schema name and the specified string array for the restriction values.

Syntax

Visual Basic

```
Public Overrides Function GetSchema (
    ByVal collection As String,
    ByVal restrictions As String()
) As DataTable
```

C#

```
public override DataTable GetSchema (
    string collection,
    string[] restrictions
)
```

Returns

A DataTable that contains schema information.

Remarks

These methods are used to query the SAP HANA database server for metadata. Each type of metadata is given a collection name, which must be passed to receive that data. The default collection name is MetaDataCollections.

You can query the data provider to determine the list of supported schema collections by calling the GetSchema method with no arguments, or with the schema collection name *MetaDataCollections*. This will return a DataTable with a list of the supported schema collections (CollectionName), the number of restrictions that they each support (NumberOfRestrictions), and the number of identifier parts that they use (NumberOfIdentifierParts).

Collection	Metadata
AllColumns	Returns information on all columns in the database.
Columns	Returns information on all columns in the database.
DataSourceInformation	Returns information about the SAP HANA database server.
DataTypes	Returns a list of supported data types.
ForeignKeys	Returns information on all foreign keys in the database.
IndexColumns	Returns information on all index columns in the database.
Indexes	Returns information on all indexes in the database.
MetaDataCollections	Returns a list of all collection names.
ProcedureParameters	Returns information on all procedure parameters in the database.
Procedures	Returns information on all procedures in the database.
ReservedWords	Returns a list of reserved words used by SAP HANA.
Restrictions	Returns information on restrictions used in GetSchema.
Tables	Returns information on all tables in the database.
Users	Returns information on all users in the database.
ViewColumns	Returns information on all columns in views in the database.

Collection	Metadata
Views	Returns information on all views in the database.

These collection names are also available as read-only properties in the HanaMetaDataCollectionNames class.

The results returned can be filtered by specifying an array of restrictions in the call to GetSchema.

The restrictions available with each collection can be queried by calling:

```
GetSchema( "Restrictions" )
```

If the collection requires four restrictions, then the restrictions parameter must be either NULL, or a string with four values.

To filter on a particular restriction, place the string to filter by in its place in the array and leave any unused places NULL. For example, the Tables collection has three restrictions: Owner, Table, and TableType.

To filter the Table collection by table_name:

```
GetSchema( "Tables", new string[ ] { NULL, "my_table", NULL } )
```

This example returns information on all tables named my_table.

```
GetSchema( "Tables", new string[ ] { "DEMO", "my_table", NULL } )
```

This example returns information on all tables named my_table owned by the schema DEMO.

The following list summarizes the columns returned by each collection. If the number of rows returned in a collection can be reduced by specifying a restriction on a column, then the restriction name for that column is shown in parentheses. The order in which restrictions are specified is the order in which they are presented in the lists below.

AllColumns collection

- TABLE_SCHEMA
- TABLE_NAME
- COLUMN_NAME
- ORDINAL_POSITION
- DATA_TYPE
- COLUMN_DEFAULT
- IS_NULLABLE
- NUMERIC_PRECISION
- NUMERIC_SCALE
- CHARACTER_MAXIMUM_LENGTH
- DATETIME_PRECISION

Columns collection

- TABLE_SCHEMA
- TABLE_NAME
- COLUMN_NAME
- ORDINAL_POSITION

- DATA_TYPE
- COLUMN_DEFAULT
- IS_NULLABLE
- NUMERIC_PRECISION
- NUMERIC_SCALE
- CHARACTER_MAXIMUM_LENGTH
- DATETIME_PRECISION

DataSourceInformation collection

- CompositeIdentifierSeparatorPattern
- DataSourceProductName
- DataSourceProductVersion
- DataSourceProductVersionNormalized
- GroupByBehavior
- IdentifierPattern
- IdentifierCase
- OrderByColumnsInSelect
- ParameterMarkerFormat
- ParameterMarkerPattern
- ParameterNameMaxLength
- ParameterNamePattern
- QuotedIdentifierPattern
- QuotedIdentifierCase
- StatementSeparatorPattern
- StringLiteralPattern
- SupportedJoinOperators

DataTypes collection

- TypeName
- ProviderDbType
- ColumnSize
- CreateFormat
- CreateParameters
- DataType
- IsAutoIncrementable
- IsBestMatch
- IsCaseSensitive
- IsFixedLength
- IsFixedPrecisionScale
- IsLong
- IsNullable
- IsSearchable
- IsSearchableWithLike

- IsUnsigned
- MaximumScale
- MinimumScale
- IsConcurrencyType
- IsLiteralSupported
- LiteralPrefix
- LiteralSuffix

ForeignKeys collection

- TABLE_SCHEMA
- TABLE_NAME
- COLUMN_NAME
- POSITION
- CONSTRAINT_NAME
- REFERENCED_SCHEMA_NAME
- REFERENCED_TABLE_NAME
- REFERENCED_COLUMN_NAME
- REFERENCED_CONSTRAINT_NAME
- UPDATE_RULE
- DELETE_RULE

IndexColumns collection

- TABLE_SCHEMA
- TABLE_NAME
- COLUMN_NAME
- POSITION
- INDEX_NAME
- CONSTRAINT
- ASCENDING_ORDER

Indexes collection

- TABLE_SCHEMA
- TABLE_NAME
- INDEX_NAME
- INDEX_TYPE
- CONSTRAINT

MetaDataCollections collection

- CollectionName
- NumberOfRestrictions
- NumberOfIdentifierParts

ProcedureParameters collection

- PROCEDURE_SCHEMA
- PROCEDURE_NAME

- PARAMETER_NAME
- DATA_TYPE_NAME
- LENGTH
- SCALE
- IS_NULLABLE
- POSITION
- PARAMETER_TYPE

Procedures collection

- ROUTINE_SCHEMA
- ROUTINE_NAME
- ROUTINE_TYPE

ReservedWords collection

- reserved_word

Restrictions collection

- CollectionName
- RestrictionName
- RestrictionDefault
- RestrictionNumber

Tables collection

- TABLE_SCHEMA
- TABLE_NAME
- TABLE_TYPE

Users collection

- USER_NAME
- USER_MODE
- CREATOR
- CREATE_TIME

ViewColumns collection

- VIEW_SCHEMA
- VIEW_NAME
- COLUMN_NAME

Views collection

- VIEW_SCHEMA
- VIEW_NAME
- VIEW_TYPE

Related Information

HanaConnection class [page 203]

4.1.4.6.15 Open() method

Opens a database connection with the property settings specified by the HanaConnection.ConnectionString.

Syntax

Visual Basic

```
Public Overrides Sub Open ()
```

C#

```
public override unsafe void Open ()
```

Related Information

ConnectionString property [page 224]
Connection Examples for the SAP HANA Data Provider for ADO.NET [page 64]
ADO.NET Connection Properties [page 41]

4.1.4.6.16 ConnectionString property

Provides the database connection string.

Syntax

Visual Basic

```
Public Overrides Property ConnectionString As String
```

C#

```
public override string ConnectionString {get;set;}
```

Remarks

The ConnectionString is designed to match the SAP HANA connection string format as closely as possible with the following exception: when the Persist Security Info connection parameter is set to false (the default), the connection string that is returned is the same as the user-specified ConnectionString minus security

information(the password). The data provider does not persist the password in a returned connection string unless you set the Persist Security Info connection parameter to true.

To encrypt the ConnectionString property, on Microsoft Windows platforms you must specify `ENCRYPT=TRUE`. For .NET Core applications on Linux or Mac, `ENCRYPT=TRUE` must be added, and either the OpenSSL or CommonCryptoLib provider must be configured to accept the DigiCert Global Root G2 certificate.

Use the ConnectionString property to connect to a variety of data sources.

You can set the ConnectionString property only when the connection is closed. Many of the connection string values have corresponding read-only properties. When the connection string is set, all of these properties are updated, unless an error is detected. If an error is detected, then none of the properties are updated. HanaConnection properties return only those settings contained in the ConnectionString.

If you reset the ConnectionString on a closed connection, then all connection string values and related properties are reset, including the password.

When the property is set, a preliminary validation of the connection string is performed. When an application calls the Open method, the connection string is fully validated. A runtime exception is generated if the connection string contains invalid or unsupported properties.

Values can be delimited by single or double quotes. Either single or double quotes may be used within a connection string by using the other delimiter, for example, name="value's" or name= 'value"s', but not name='value's' or name= ""value"". Blank characters are ignored unless they are placed within a value or within quotes. keyword=value pairs must be separated by a semicolon. If a semicolon is part of a value, then it must also be delimited by quotes. Escape sequences are not supported, and the value type is irrelevant. Names are not case sensitive. If a property name occurs more than once in the connection string, then the value associated with the last occurrence is used.

Use caution when constructing a connection string based on user input, such as when retrieving a user ID and password from a window, and appending it to the connection string. The application should not allow a user to embed extra connection string parameters in these values.

The default value of the pooling connection parameter is true (pooling=true).

Example

The following statements specify a connection string and open the connection.

```
HanaConnection conn = new HanaConnection();
conn.ConnectionString = "server=hana-db:30015;userid=JSmith;password=secret";
conn.Open();
```

Related Information

SAP Note 3397584
SAP Note 3327214
SAP Note 3399572
HanaConnection class [page 203]

Open() method [page 224]
ADO.NET Connection Properties [page 41]
Secure Communication Between SAP HANA Cloud and JDBC/ODBC Clients
Client-Side TLS Connection Properties (ODBC)

4.1.4.6.17 ConnectionTimeout property

Gets the number of seconds before a connection attempt times out with an error.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property ConnectionTimeout As Integer
```

C#

```
public override int ConnectionTimeout {get;}
```

Remarks

The default ConnectionTimeout value is 15 seconds.

Example

The following statement displays the value of the ConnectionTimeout.

```
MessageBox.Show( conn.ConnectionTimeout.ToString( ) );
```

4.1.4.6.18 Credential property

Gets or sets the HanaCredential object for this connection.

Syntax

Visual Basic

```
Public Property Credential As HanaCredential
```

C#

```
public HanaCredential Credential {get;set;}
```

Returns

The HanaCredential object for this connection.

Related Information

HanaConnection class [page 203]
Open() method [page 224]

4.1.4.6.19 Database property

Gets the SAP HANA database server name (for example, GA1).

Syntax

Visual Basic

```
Public ReadOnly Overrides Property Database As String
```

C#

```
public override string Database {get;}
```

Remarks

If the connection is open, then the HanaConnection object returns the name of the current database. Otherwise, the HanaConnection object looks in the connection string for the Database connection parameter.

4.1.4.6.20 DataSource property

Gets the SAP HANA database server host name (for example, hana-db).

Syntax

Visual Basic

```
Public ReadOnly Overrides Property DataSource As String
```

C#

```
public override string DataSource {get;}
```

Remarks

If the connection is open, then the HanaConnection object returns the host name. Otherwise, the HanaConnection object looks in the connection string for the Server connection parameter.

Related Information

HanaConnection class [page 203]

4.1.4.6.21 InitString property

A SQL statement that is executed immediately after the connection is established.

Syntax

Visual Basic

```
Public Property InitString As String
```

C#

```
public string InitString {get;set;}
```

Remarks

The InitString is executed immediately after the connection is opened.

4.1.4.6.22 ServerVersion property

Gets a string that contains the version of the instance of SAP HANA to which the client is connected.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property ServerVersion As String
```

C#

```
public override string ServerVersion {get;}
```

Returns

The database server version string.

Remarks

The database server version string has the following format: ##.##.##.#### (major.minor.patch.build).

4.1.4.6.23 State property

Indicates the state of the HanaConnection object.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property State As ConnectionState
```

C#

```
public override ConnectionState State {get;}
```

Returns

A System.Data.ConnectionState enumeration.

4.1.4.6.24 InfoMessage event

Occurs when the SAP HANA database server returns a warning or informational message.

Syntax

Visual Basic

```
Public Event InfoMessage  As HanaInfoMessageEventHandler
```

C#

```
public HanaInfoMessageEventHandler InfoMessage;
```

Remarks

The event handler receives an argument of type HanaInfoMessageEventArgs containing data related to this event. The following HanaInfoMessageEventArgs properties provide information specific to this event: NativeError, Errors, Message, MessageType, and Source.

For more information, see the Microsoft .NET Framework documentation for OleDbConnection.InfoMessage Event.

4.1.4.6.25 StateChange event

Occurs when the state of the HanaConnection object changes.

Syntax

Visual Basic

```
Public Event StateChange  As StateChangeEventHandler
```

C#

```
public override StateChangeEventHandler StateChange;
```

Remarks

The event handler receives an argument of type StateChangeEventArgs with data related to this event. The following StateChangeEventArgs properties provide information specific to this event: CurrentState and OriginalState.

For more information, see the Microsoft .NET Framework documentation for OleDbConnection.StateChange Event.

4.1.4.7 HanaConnectionStringBuilder class

Provides a simple way to create and manage the contents of connection strings used by the HanaConnection class.

Syntax

Visual Basic

```
Public NotInheritable Class HanaConnectionStringBuilder Inherits
HanaConnectionStringBuilderBase
```

C#

```
public sealed class HanaConnectionStringBuilder :
HanaConnectionStringBuilderBase
```

Members

All members of HanaConnectionStringBuilder, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaConnectionStringBuilder [page 233]	Initializes a new instance of the HanaConnectionStringBuilder class.

Properties

Modifier and Type	Property	Default Value	Description
public int	ConnectionLifetime [page 234]	0	Gets or sets the ConnectionLifetime connection property.
public int	ConnectionTimeout [page 234]	15	Gets or sets the ConnectionTimeout connection property.
public string	CurrentSchema [page 235]	`<empty-string>`	Gets or sets the currentSchema connection property.
public string	Database [page 235]	`<empty-string>`	Gets or sets the database connection property.
public string	DataSourceName [page 236]	`<empty-string>`	Gets or sets the DataSourceName connection property.
public string	InitString [page 236]	`<empty-string>`	Gets or sets the InitString connection property.
public string	Integrated [page 236]	No	
public string	IsolationLevel [page 237]	`<empty-string>`	Gets or sets the IsolationLevel connection property.
public int	MaxPoolSize [page 237]	100	Gets or sets the MaxPoolSize connection property.
public int	MinPoolSize [page 237]	0	Gets or sets the MinPoolSize connection property.
public string	Password [page 238]	`<empty-string>`	Gets or sets the Password connection property.
public bool	PersistSecurityInfo [page 238]	FALSE	Gets or sets the PersistSecurityInfo connection property.
public bool	Pooling [page 238]	TRUE	Gets or sets the Pooling connection property.
public string	Server [page 239]	`<empty-string>`	Gets or sets the Server connection property.

Modifier and Type	Property	Default Value	Description
public string	UserName [page 239]	`<empty-string>`	Gets or sets the UserName connection property.

Inherited members from HanaConnectionStringBuilderBase

Modifier and Type	Member	Description
public override bool	ContainsKey(string) [page 240]	Determines whether the HanaConnectionStringBuilder object contains a specific keyword.
public string	GetKeyword(string) [page 241]	Gets the keyword for the specified HanaConnectionStringBuilder property.
public bool	GetUseLongNameAsKeyword() [page 242]	Gets a boolean value that indicates whether long connection parameter names are used in the connection string.
public override ICollection	Keys [page 245]	Gets a System.Collections.ICollection that contains the keys in the HanaConnectionStringBuilder.
public override bool	Remove(string) [page 242]	Removes the entry with the specified key from the HanaConnectionStringBuilder instance.
public void	SetUseLongNameAsKeyword(bool) [page 243]	Sets a boolean value that indicates whether long connection parameter names are used in the connection string.
public override bool	ShouldSerialize(string) [page 244]	Indicates whether the specified key exists in this HanaConnectionStringBuilder instance.
public override object	this[string keyword] [page 245]	Gets or sets the value of the connection keyword.
public override bool	TryGetValue(string, out object) [page 244]	Retrieves a value corresponding to the supplied key from this HanaConnectionStringBuilder.

Remarks

The HanaConnectionStringBuilder class inherits HanaConnectionStringBuilderBase, which inherits DbConnectionStringBuilder.

For SAP HANA Cloud, encrypting connection strings for versions 2.5 and lower must be set explicitly. For SAP HANA Cloud versions 2.6 and higher, encryption is auto-handled.

For a list of connection parameters, see "ADO.NET Connection Properties".

Related Information

ADO.NET Connection Properties [page 41]

4.1.4.7.1 HanaConnectionStringBuilder constructor

Initializes a new instance of the HanaConnectionStringBuilder class.

Overload list

Modifier and Type	Overload name	Description
public	HanaConnectionStringBuilder() [page 233]	Initializes a new instance of the HanaConnectionStringBuilder class.
public	HanaConnectionStringBuilder(string) [page 233]	Initializes a new instance of the HanaConnectionStringBuilder class.

4.1.4.7.1.1 HanaConnectionStringBuilder() constructor

Initializes a new instance of the HanaConnectionStringBuilder class.

Syntax

Visual Basic

```
Public Sub HanaConnectionStringBuilder ()
```

C#

```
public HanaConnectionStringBuilder ()
```

4.1.4.7.1.2 HanaConnectionStringBuilder(string) constructor

Initializes a new instance of the HanaConnectionStringBuilder class.

Syntax

Visual Basic

```
Public Sub HanaConnectionStringBuilder (ByVal connectionString As
String)
```

C#

```
public HanaConnectionStringBuilder (string connectionString)
```

Parameters

connectionString

The basis for the object's internal connection information. Parsed into keyword=value pairs. For a list of connection parameters, see Connection parameters.

Example

The following statement initializes a HanaConnection object for a connection to an SAP HANA database server named HumanResources. The connection uses the user ID Admin and the password Money123.

```
HanaConnectionStringBuilder sb = new HanaConnectionStringBuilder(
 "UserID=Admin;Password=Money123;Server=HumanResources:30015" );
```

4.1.4.7.2 ConnectionLifetime property

Gets or sets the ConnectionLifetime connection property.

Syntax

Visual Basic

```
Public Property ConnectionLifetime As Integer
```

C#

```
public int ConnectionLifetime {get;set;}
```

4.1.4.7.3 ConnectionTimeout property

Gets or sets the ConnectionTimeout connection property.

Syntax

Visual Basic

```
Public Property ConnectionTimeout As Integer
```

C#

```
public int ConnectionTimeout {get;set;}
```

Example

The following statement displays the value of the ConnectionTimeout property.

```
MessageBox.Show( connString.ConnectionTimeout.ToString() );
```

4.1.4.7.4 CurrentSchema property

Gets or sets the currentSchema connection property.

Syntax

Visual Basic

```
Public Property CurrentSchema As String
```

C#

```
public string CurrentSchema {get;set;}
```

4.1.4.7.5 Database property

Gets or sets the Database connection property.

Syntax

Visual Basic

```
Public Property Database As String
```

C#

```
public string Database {get;set;}
```

4.1.4.7.6 DataSourceName property

Gets or sets the DataSourceName connection property.

Syntax

Visual Basic

```
Public Property DataSourceName As String
```

C#

```
public string DataSourceName {get;set;}
```

4.1.4.7.7 InitString property

Gets or sets the InitString connection property.

Syntax

Visual Basic

```
Public Property InitString As String
```

C#

```
public string InitString {get;set;}
```

4.1.4.7.8 Integrated property

Syntax

Visual Basic

```
Public Property Integrated As String
```

C#

```
public string Integrated {get;set;}
```

4.1.4.7.9 IsolationLevel property

Gets or sets the IsolationLevel connection property.

Syntax

Visual Basic

```
Public Property IsolationLevel As String
```

C#

```
public string IsolationLevel {get;set;}
```

4.1.4.7.10 MaxPoolSize property

Gets or sets the MaxPoolSize connection property.

Syntax

Visual Basic

```
Public Property MaxPoolSize As Integer
```

C#

```
public int MaxPoolSize {get;set;}
```

4.1.4.7.11 MinPoolSize property

Gets or sets the MinPoolSize connection property.

Syntax

Visual Basic

```
Public Property MinPoolSize As Integer
```

C#

```
public int MinPoolSize {get;set;}
```

4.1.4.7.12 Password property

Gets or sets the Password connection property.

Syntax

Visual Basic

```
Public Property Password As String
```

C#

```
public string Password {get;set;}
```

4.1.4.7.13 PersistSecurityInfo property

Gets or sets the PersistSecurityInfo connection property.

Syntax

Visual Basic

```
Public Property PersistSecurityInfo As Boolean
```

C#

```
public bool PersistSecurityInfo {get;set;}
```

4.1.4.7.14 Pooling property

Gets or sets the Pooling connection property.

Syntax

Visual Basic

```
Public Property Pooling As Boolean
```

C#

```
public bool Pooling {get;set;}
```

4.1.4.7.15 Server property

Gets or sets the Server connection property.

Syntax

Visual Basic

```
Public Property Server As String
```

C#

```
public string Server {get;set;}
```

4.1.4.7.16 UserName property

Gets or sets the UserName connection property.

Syntax

Visual Basic

```
Public Property UserName As String
```

C#

```
public string UserName {get;set;}
```

4.1.4.8 HanaConnectionStringBuilderBase class

Base class of the HanaConnectionStringBuilder class.

Syntax

Visual Basic

```
Public MustInherit Class HanaConnectionStringBuilderBase Inherits
System.Data.Common.DbConnectionStringBuilder
```

C#

```
public abstract class HanaConnectionStringBuilderBase :
System.Data.Common.DbConnectionStringBuilder
```

Members

All members of HanaConnectionStringBuilderBase, including inherited members.

Methods

Modifier and Type	Method	Description
public override bool	ContainsKey(string) [page 240]	Determines whether the HanaConnectionStringBuilder object contains a specific keyword.
public string	GetKeyword(string) [page 241]	Gets the keyword for the specified HanaConnectionStringBuilder property.
public bool	GetUseLongNameAsKeyword() [page 242]	Gets a boolean value that indicates whether long connection parameter names are used in the connection string.
public override bool	Remove(string) [page 242]	Removes the entry with the specified key from the HanaConnectionStringBuilder instance.
public void	SetUseLongNameAsKeyword(bool) [page 243]	Sets a boolean value that indicates whether long connection parameter names are used in the connection string.
public override bool	ShouldSerialize(string) [page 244]	Indicates whether the specified key exists in this HanaConnectionStringBuilder instance.
public override bool	TryGetValue(string, out object) [page 244]	Retrieves a value corresponding to the supplied key from this HanaConnectionStringBuilder.

Properties

Modifier and Type	Property	Description
public override ICollection	Keys [page 245]	Gets a System.Collections.ICollection that contains the keys in the HanaConnectionStringBuilder.
public override object	this[string keyword] [page 245]	Gets or sets the value of the connection keyword.

4.1.4.8.1 ContainsKey(string) method

Determines whether the HanaConnectionStringBuilder object contains a specific keyword.

Syntax

Visual Basic

```
Public Overrides Function ContainsKey (ByVal keyword As String) As Boolean
```

C#

```
public override bool ContainsKey (string keyword)
```

Parameters

keyword

The keyword to locate in the HanaConnectionStringBuilder.

Returns

True if the value associated with keyword has been set; false otherwise.

Example

The following statement determines whether the HanaConnectionStringBuilder object contains the UserID keyword.

```
connectString.ContainsKey("UserID")
```

4.1.4.8.2 GetKeyword(string) method

Gets the keyword for the specified HanaConnectionStringBuilder property.

Syntax

Visual Basic

```
Public Function GetKeyword (ByVal propName As String) As String
```

C#

```
public string GetKeyword (string propName)
```

Parameters

propName

The name of the HanaConnectionStringBuilder property.

Returns

The keyword for the specified HanaConnectionStringBuilder property.

4.1.4.8.3 GetUseLongNameAsKeyword() method

Gets a boolean value that indicates whether long connection parameter names are used in the connection string.

Syntax

Visual Basic

```
Public Function GetUseLongNameAsKeyword () As Boolean
```

C#

```
public bool GetUseLongNameAsKeyword ()
```

Returns

True if long connection parameter names are used to build connection strings; false otherwise.

Remarks

SAP HANA connection parameters have both long and short forms of their names. For example, to specify a password in your connection string, you can use either of the following parameters: Password or PWD. By default, long connection parameter names are used to build connection strings.

Related Information

SetUseLongNameAsKeyword(bool) method [page 243]

4.1.4.8.4 Remove(string) method

Removes the entry with the specified key from the HanaConnectionStringBuilder instance.

Syntax

Visual Basic

```
Public Overrides Function Remove (ByVal keyword As String) As
Boolean
```

C#

```
public override bool Remove (string keyword)
```

Parameters

keyword

The key of the key/value pair to be removed from the connection string in this HanaConnectionStringBuilder.

Returns

True if the key existed within the connection string and was removed; false if the key did not exist.

4.1.4.8.5 SetUseLongNameAsKeyword(bool) method

Sets a boolean value that indicates whether long connection parameter names are used in the connection string.

Syntax

Visual Basic

```
Public Sub SetUseLongNameAsKeyword (ByVal useLongNameAsKeyword As Boolean)
```

C#

```
public void SetUseLongNameAsKeyword (bool useLongNameAsKeyword)
```

Parameters

useLongNameAsKeyword

A boolean value that indicates whether the long connection parameter name is used in the connection string.

Remarks

Long connection parameter names are used by default.

Related Information

GetUseLongNameAsKeyword() method [page 242]

4.1.4.8.6 ShouldSerialize(string) method

Indicates whether the specified key exists in this HanaConnectionStringBuilder instance.

Syntax

Visual Basic

```
Public Overrides Function ShouldSerialize (ByVal keyword As
String) As Boolean
```

C#

```
public override bool ShouldSerialize (string keyword)
```

Parameters

keyword

The key to locate in the HanaConnectionStringBuilder.

Returns

True if the HanaConnectionStringBuilder contains an entry with the specified key; false otherwise.

4.1.4.8.7 TryGetValue(string, out object) method

Retrieves a value corresponding to the supplied key from this HanaConnectionStringBuilder.

Syntax

Visual Basic

```
Public Overrides Function TryGetValue (
    ByVal keyword As String,
    ByVal value As Object
) As Boolean
```

C#

```
public override bool TryGetValue (
    string keyword,
    out object value
)
```

Parameters

keyword

The key of the item to retrieve.

value

The value corresponding to keyword.

Returns

True if keyword was found within the connection string; false otherwise.

4.1.4.8.8 Keys property

Gets a System.Collections.ICollection that contains the keys in the HanaConnectionStringBuilder.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property Keys As ICollection
```

C#

```
public override ICollection Keys {get;}
```

Returns

A System.Collections.ICollection that contains the keys in the HanaConnectionStringBuilder.

4.1.4.8.9 this[string keyword] property

Gets or sets the value of the connection keyword.

Syntax

Visual Basic

```
Public Overrides Property Item (ByVal keyword As String) As Object
```

C#

```
public override object this[string keyword] {get;set;}
```

Remarks

```
HanaConnectionStringBuilder sb = new HanaConnectionStringBuilder();
sb["Password"] = "secret";
```

An object representing the value of the specified connection keyword.

If the keyword or type is invalid, then an exception is raised. The parameter value is case insensitive.

When setting the value, passing NULL clears the value.

4.1.4.9 HanaCredential class

HanaCredential provides a more secure way to specify the password for a login attempt using SAP HANA authentication. This class is not supported for .NET Core.

Syntax

Visual Basic

```
Public NotInheritable Class HanaCredential
```

C#

```
public sealed class HanaCredential
```

Members

All members of HanaCredential, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaCredential(string, SecureString) [page 247]	Initializes a HanaCredential object.

Properties

Modifier and Type	Property	Description
public SecureString	Password [page 248]	Returns the password of the HanaCredential object.
public string	UserId [page 248]	Returns the user ID of the HanaCredential object.

Remarks

HanaCredential is comprised of a user ID and a password that are used for SAP HANA Authentication. The password in a HanaCredential object is of type System.Security.SecureString. HanaCredential cannot be inherited.

4.1.4.9.1 HanaCredential(string, SecureString) constructor

Initializes a HanaCredential object. This constructor is not supported for .NET Core.

Syntax

Visual Basic

```
Public Sub HanaCredential (
    ByVal userId As String,
    ByVal password As SecureString
)
```

C#

```
public HanaCredential (
    string userId,
    SecureString password
)
```

Parameters

userId

The user ID.

password

The password; a System.Security.SecureString value marked as read-only. Passing a read/write System.Security.SecureString parameter will raise a System.ArgumentException.

Related Information

HanaConnection class [page 203]

4.1.4.9.2 Password property

Returns the password of the HanaCredential object. This property is not supported for .NET Core.

Syntax

Visual Basic

```
Public ReadOnly Property Password As SecureString
```

C#

```
public SecureString Password {get;}
```

Returns

The password of the HanaCredential object.

4.1.4.9.3 UserId property

Returns the user ID of the HanaCredential object. This property is not supported for .NET Core.

Syntax

Visual Basic

```
Public ReadOnly Property UserId As String
```

C#

```
public string UserId {get;}
```

Returns

The user ID of the HanaCredential object.

4.1.4.10 HanaDataAdapter class

Represents a set of commands and a database connection used to fill a System.Data.DataSet and to update a database.

Syntax

Visual Basic

```
Public NotInheritable Class HanaDataAdapter Inherits
System.Data.Common.DbDataAdapter Implements System.ICloneable
```

C#

```
public sealed class HanaDataAdapter :
System.Data.Common.DbDataAdapter, System.ICloneable
```

Members

All members of HanaDataAdapter, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaDataAdapter [page 251]	Initializes a HanaDataAdapter object.

Methods

Modifier and Type	Method	Description
protected override void	ClearBatch() [page 254]	Removes all HanaCommand objects from the batch.
protected override RowUpdatedEventArgs	CreateRowUpdatedEvent(DataRow, IDbCommand, StatementType, DataTableMapping) [page 255]	Initializes a new instance of the System.Data.Common.RowUpdatedEventArgs class.
protected override RowUpdatingEventArgs	CreateRowUpdatingEvent(DataRow, IDbCommand, StatementType, DataTableMapping) [page 256]	Initializes a new instance of the System.Data.Common.RowUpdatingEventArgs class.
protected override void	Dispose(bool) [page 257]	Releases the unmanaged resources used by the HanaDataAdapter object and optionally releases the managed resources.
protected override int	Fill [page 257]	Adds or refreshes rows in a System.Data.DataSet or System.Data.DataTable object with data from the database.

Modifier and Type	Method	Description
protected override DataTable[]	FillSchema [page 261]	Adds a System.Data.DataTable to a System.Data.DataSet and configures the schema to match the schema in the data source.
public new HanaParameter[]	GetFillParameters() [page 263]	Returns the parameters set by you when executing a SELECT statement.
protected override void	InitializeBatching() [page 264]	Initializes batching for the HanaDataAdapter object.
protected override void	OnRowUpdated(RowUpdatedEventArgs) [page 264]	Raises the RowUpdated event of a data provider.
protected override void	OnRowUpdating(RowUpdatingEventArgs) [page 265]	Raises the RowUpdating event of a data provider.
protected override void	TerminateBatching() [page 265]	Ends batching for the HanaDataAdapter object.
protected override int	Update(DataRow[], DataTableMapping) [page 266]	Updates the tables in a database with the changes made to the DataSet.

Properties

Modifier and Type	Property	Description
public new HanaCommand	DeleteCommand [page 267]	Specifies a HanaCommand object that is executed against the database when the Update method is called to delete rows in the database that correspond to deleted rows in the DataSet.
public new HanaCommand	InsertCommand [page 267]	Specifies a HanaCommand that is executed against the database when the Update method is called that adds rows to the database to correspond to rows that were inserted in the DataSet.
public new HanaCommand	SelectCommand [page 268]	Specifies a HanaCommand that is used during Fill or FillSchema to obtain a result set from the database for copying into a DataSet.
public new DataTableMappingCollection	TableMappings [page 268]	Specifies a collection that provides the master mapping between a source table and a DataTable.
public override int	UpdateBatchSize [page 269]	Gets or sets the number of rows that are processed in each round-trip to the SAP HANA database server.
public new HanaCommand	UpdateCommand [page 270]	Specifies a HanaCommand that is executed against the database when the Update method is called to update rows in the database that correspond to updated rows in the DataSet.

Events

Modifier and Type	Event	Description
public HanaRowUpdatedEventHandler	RowUpdated [page 270]	Occurs during an update after a command is executed against the data source.
public HanaRowUpdatingEventHandler	RowUpdating [page 271]	Occurs during an update before a command is executed against the data source.

Remarks

The System.Data.DataSet provides a way to work with data offline. The HanaDataAdapter provides methods to associate a DataSet with a set of SQL statements.

Implements: IDbDataAdapter, IDataAdapter, ICloneable

For more information, see HanaDataAdapter overview and Data access and manipulation.

4.1.4.10.1 HanaDataAdapter constructor

Initializes a HanaDataAdapter object.

Overload list

Modifier and Type	Overload name	Description
public	HanaDataAdapter() [page 252]	Initializes a HanaDataAdapter object.
public	HanaDataAdapter(HanaCommand) [page 252]	Initializes a HanaDataAdapter object with the specified SELECT statement.
public	HanaDataAdapter(string, HanaConnection) [page 253]	Initializes a HanaDataAdapter object with the specified SELECT statement and connection.
public	HanaDataAdapter(string, string) [page 254]	Initializes a HanaDataAdapter object with the specified SELECT statement and connection string.

4.1.4.10.1.1 HanaDataAdapter() constructor

Initializes a HanaDataAdapter object.

Syntax

Visual Basic

```
Public Sub HanaDataAdapter ()
```

C#

```
public HanaDataAdapter ()
```

Related Information

HanaDataAdapter(HanaCommand) constructor [page 252]
HanaDataAdapter(string, HanaConnection) constructor [page 253]
HanaDataAdapter(string, string) constructor [page 254]

4.1.4.10.1.2 HanaDataAdapter(HanaCommand) constructor

Initializes a HanaDataAdapter object with the specified SELECT statement.

Syntax

Visual Basic

```
Public Sub HanaDataAdapter (ByVal selectCommand As HanaCommand)
```

C#

```
public HanaDataAdapter (HanaCommand selectCommand)
```

Parameters

selectCommand

A HanaCommand object that is used during System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet) to select records from the data source for placement in the System.Data.DataSet.

Related Information

HanaDataAdapter() constructor [page 252]

HanaDataAdapter(string, HanaConnection) constructor [page 253]
HanaDataAdapter(string, string) constructor [page 254]

4.1.4.10.1.3 HanaDataAdapter(string, HanaConnection) constructor

Initializes a HanaDataAdapter object with the specified SELECT statement and connection.

Syntax

Visual Basic

```
Public Sub HanaDataAdapter (
    ByVal selectCommandText As String,
    ByVal selectConnection As HanaConnection
)
```

C#

```
public HanaDataAdapter (
    string selectCommandText,
    HanaConnection selectConnection
)
```

Parameters

selectCommandText

A SELECT statement to be used to set the HanaDataAdapter.SelectCommand property of the HanaDataAdapter object.

selectConnection

A HanaConnection object that defines a connection to an SAP HANA database server.

Related Information

HanaDataAdapter() constructor [page 252]
HanaDataAdapter(HanaCommand) constructor [page 252]
HanaDataAdapter(string, string) constructor [page 254]
SelectCommand property [page 268]
HanaConnection class [page 203]

4.1.4.10.1.4 HanaDataAdapter(string, string) constructor

Initializes a HanaDataAdapter object with the specified SELECT statement and connection string.

Syntax

Visual Basic

```
Public Sub HanaDataAdapter (
    ByVal selectCommandText As String,
    ByVal selectConnectionString As String
)
```

C#

```
public HanaDataAdapter (
    string selectCommandText,
    string selectConnectionString
)
```

Parameters

selectCommandText

A SELECT statement to be used to set the HanaDataAdapter.SelectCommand property of the HanaDataAdapter object.

selectConnectionString

A connection string for an SAP HANA database server.

Related Information

HanaDataAdapter() constructor [page 252]
HanaDataAdapter(HanaCommand) constructor [page 252]
HanaDataAdapter(string, HanaConnection) constructor [page 253]
SelectCommand property [page 268]

4.1.4.10.2 ClearBatch() method

Removes all HanaCommand objects from the batch.

Syntax

Visual Basic

```
Protected Overrides Sub ClearBatch ()
```

C#

```
protected override void ClearBatch ()
```

Related Information

HanaCommand class [page 147]

4.1.4.10.3 CreateRowUpdatedEvent(DataRow, IDbCommand, StatementType, DataTableMapping) method

Initializes a new instance of the System.Data.Common.RowUpdatedEventArgs class.

Syntax

Visual Basic

```
Protected Overrides Function CreateRowUpdatedEvent (
    ByVal dataRow As DataRow,
    ByVal command As IDbCommand,
    ByVal statementType As StatementType,
    ByVal tableMapping As DataTableMapping
) As RowUpdatedEventArgs
```

C#

```
protected override RowUpdatedEventArgs CreateRowUpdatedEvent (
    DataRow dataRow,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
)
```

Parameters

dataRow

The System.Data.DataRow used to update the data source.

command

The System.Data.IDbCommand executed during the System.Data.IDataAdapter.Update(System.Data.DataSet).

statementType

Whether the command is an UPDATE, INSERT, DELETE, or SELECT statement.

tableMapping

A System.Data.Common.DataTableMapping object.

Returns

A new instance of the System.Data.Common.RowUpdatedEventArgs class.

4.1.4.10.4 CreateRowUpdatingEvent(DataRow, IDbCommand, StatementType, DataTableMapping) method

Initializes a new instance of the System.Data.Common.RowUpdatingEventArgs class.

Syntax

Visual Basic

```
Protected Overrides Function CreateRowUpdatingEvent (
    ByVal dataRow As DataRow,
    ByVal command As IDbCommand,
    ByVal statementType As StatementType,
    ByVal tableMapping As DataTableMapping
) As RowUpdatingEventArgs
```

C#

```
protected override RowUpdatingEventArgs CreateRowUpdatingEvent (
    DataRow dataRow,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
)
```

Parameters

dataRow

The System.Data.DataRow used to update the data source.

command

The System.Data.IDbCommand executed during the System.Data.IDataAdapter.Update(System.Data.DataSet).

statementType

Whether the command is an UPDATE, INSERT, DELETE, or SELECT statement.

tableMapping

A System.Data.Common.DataTableMapping object.

Returns

A new instance of the System.Data.Common.RowUpdatingEventArgs class.

4.1.4.10.5 Dispose(bool) method

Releases the unmanaged resources used by the HanaDataAdapter object and optionally releases the managed resources.

Syntax

Visual Basic

```
Protected Overrides Sub Dispose (ByVal disposing As Boolean)
```

C#

```
protected override void Dispose (bool disposing)
```

Parameters

disposing

True releases both managed and unmanaged resources; false releases only unmanaged resources.

Related Information

HanaDataAdapter class [page 249]

4.1.4.10.6 Fill method

Adds or refreshes rows in a System.Data.DataSet or System.Data.DataTable object with data from the database.

Overload list

Modifier and Type	Overload name	Description
protected override int	Fill(DataSet, int, int, string, IDbCommand, CommandBehavior) [page 258]	Adds or refreshes rows in a System.Data.DataSet or System.Data.DataTable object with data from the database.

Modifier and Type	Overload name	Description
protected override int	Fill(DataTable[], int, int, IDbCommand, CommandBehavior) [page 259]	Adds or refreshes rows in a specified range in the System.Data.DataSet to match those in the data source using the System.Data.DataSet and System.Data.DataTable names.

4.1.4.10.6.1 Fill(DataSet, int, int, string, IDbCommand, CommandBehavior) method

Adds or refreshes rows in a System.Data.DataSet or System.Data.DataTable object with data from the database.

Syntax

Visual Basic

```
Protected Overrides Function Fill (
    ByVal dataSet As DataSet,
    ByVal startRecord As Integer,
    ByVal maxRecords As Integer,
    ByVal srcTable As String,
    ByVal command As IDbCommand,
    ByVal behavior As CommandBehavior
) As Integer
```

C#

```
protected override int Fill (
    DataSet dataSet,
    int startRecord,
    int maxRecords,
    string srcTable,
    IDbCommand command,
    CommandBehavior behavior
)
```

Parameters

dataSet

A System.Data.DataSet to fill with records and optionally, schema.

startRecord

The zero-based record number with which to start.

maxRecords

The maximum number of records to be read into the System.Data.DataSet.

srcTable

The name of the source table to use for table mapping.

command

The SQL SELECT statement used to retrieve rows from the data source.

behavior

One of the CommandBehavior values.

Returns

The number of rows successfully added or refreshed in the System.Data.DataSet.

Remarks

Even if you use the startRecord argument to limit the number of records that are copied to the DataSet, all records in the HanaDataAdapter query are fetched from the database to the client. For large result sets, this can have a significant performance impact.

An alternative is to use a HanaDataReader when a read-only, forward-only result set is sufficient, perhaps with SQL statements (ExecuteNonQuery) to undertake modifications. Another alternative is to write a stored procedure that returns only the result you need.

If SelectCommand does not return any rows, then no tables are added to the DataSet and no exception is raised.

For more information, see HanaDataAdapter overview.

4.1.4.10.6.2 Fill(DataTable[], int, int, IDbCommand, CommandBehavior) method

Adds or refreshes rows in a specified range in the System.Data.DataSet to match those in the data source using the System.Data.DataSet and System.Data.DataTable names.

Syntax

Visual Basic

```
Protected Overrides Function Fill (
    ByVal dataTables As DataTable(),
    ByVal startRecord As Integer,
    ByVal maxRecords As Integer,
    ByVal command As IDbCommand,
    ByVal behavior As CommandBehavior
) As Integer
```

C#

```
protected override int Fill (
    DataTable[] dataTables,
    int startRecord,
    int maxRecords,
    IDbCommand command,
    CommandBehavior behavior
```

```
                    )
```

Parameters

dataTables

The System.Data.DataTable objects to fill from the data source.

startRecord

The zero-based record number to start with.

maxRecords

The maximum number of records to retrieve.

command

The System.Data.IDbCommand executed to fill the System.Data.DataTable objects.

behavior

One of the System.Data.CommandBehavior values.

Returns

The number of rows added to or refreshed in the data tables.

Remarks

Even if you use the startRecord argument to limit the number of records that are copied to the DataSet, all records in the HanaDataAdapter query are fetched from the database to the client. For large result sets, this can have a significant performance impact.

An alternative is to use a HanaDataReader when a read-only, forward-only result set is sufficient, perhaps with SQL statements (ExecuteNonQuery) to undertake modifications. Another alternative is to write a stored procedure that returns only the result you need.

If SelectCommand does not return any rows, then no tables are added to the DataSet and no exception is raised.

For more information, see HanaDataAdapter overview.

4.1.4.10.7 FillSchema method

Adds a System.Data.DataTable to a System.Data.DataSet and configures the schema to match the schema in the data source.

Overload list

Modifier and Type	Overload name	Description
protected override DataTable[]	FillSchema(DataSet, SchemaType, IDbCommand, string, CommandBehavior) [page 261]	Adds a System.Data.DataTable to a System.Data.DataSet and configures the schema to match the schema in the data source.
protected override DataTable	FillSchema(DataTable, SchemaType, IDbCommand, CommandBehavior) [page 262]	Adds a System.Data.DataTable to a System.Data.DataSet and configures the schema to match the schema in the data source.

4.1.4.10.7.1 FillSchema(DataSet, SchemaType, IDbCommand, string, CommandBehavior) method

Adds a System.Data.DataTable to a System.Data.DataSet and configures the schema to match the schema in the data source.

Syntax

Visual Basic

```
Protected Overrides Function FillSchema (
    ByVal dataSet As DataSet,
    ByVal schemaType As SchemaType,
    ByVal command As IDbCommand,
    ByVal srcTable As String,
    ByVal behavior As CommandBehavior
) As DataTable()
```

C#

```
protected override DataTable[] FillSchema (
    DataSet dataSet,
    SchemaType schemaType,
    IDbCommand command,
    string srcTable,
    CommandBehavior behavior
)
```

Parameters

dataSet

A System.Data.DataSet to fill with the schema.

schemaType

One of the System.Data.SchemaType values that specify how to insert the schema.

command

The SQL SELECT statement used to retrieve rows from the data source.

srcTable

The name of the source table to use for table mapping.

behavior

One of the System.Data.CommandBehavior values.

Returns

A reference to a collection of System.Data.DataTable objects that were added to the System.Data.DataSet.

Remarks

For more information, see System.Data.IDataAdapter.FillSchema and HanaDataAdapter: Format a DataTable using FillSchema.

4.1.4.10.7.2 FillSchema(DataTable, SchemaType, IDbCommand, CommandBehavior) method

Adds a System.Data.DataTable to a System.Data.DataSet and configures the schema to match the schema in the data source.

Syntax

Visual Basic

```
Protected Overrides Function FillSchema (
    ByVal dataTable As DataTable,
    ByVal schemaType As SchemaType,
    ByVal command As IDbCommand,
    ByVal behavior As CommandBehavior
) As DataTable
```

C#

```
protected override DataTable FillSchema (
    DataTable dataTable,
    SchemaType schemaType,
```

```
    IDbCommand command,
    CommandBehavior behavior
)
```

Parameters

dataTable

A System.Data.DataTable to fill with the schema.

schemaType

One of the System.Data.SchemaType values that specify how to insert the schema.

command

The SQL SELECT statement used to retrieve rows from the data source.

behavior

One of the System.Data.CommandBehavior values.

Returns

A reference to the System.Data.DataTable object that contains the schema.

Remarks

For more information, see System.Data.Common.DbDataAdapter.FillSchema(System.Data.DataTable,System.Data.SchemaType) and HanaDataAdapter: Format a DataTable using FillSchema.

4.1.4.10.8 GetFillParameters() method

Returns the parameters set by you when executing a SELECT statement.

Syntax

Visual Basic

```
Public Shadows Function GetFillParameters () As HanaParameter()
```

C#

```
public new HanaParameter[] GetFillParameters ()
```

Returns

An array of IDataParameter objects that contains the parameters set by the user.

4.1.4.10.9 InitializeBatching() method

Initializes batching for the HanaDataAdapter object.

Syntax

Visual Basic

```
Protected Overrides Sub InitializeBatching ()
```

C#

```
protected override void InitializeBatching ()
```

Related Information

HanaDataAdapter class [page 249]

4.1.4.10.10 OnRowUpdated(RowUpdatedEventArgs) method

Raises the RowUpdated event of a data provider.

Syntax

Visual Basic

```
Protected Overrides Sub OnRowUpdated (ByVal value As
RowUpdatedEventArgs)
```

C#

```
protected override void OnRowUpdated (RowUpdatedEventArgs value)
```

Parameters

value

A System.Data.Common.RowUpdatedEventArgs that contains the event data.

4.1.4.10.11 OnRowUpdating(RowUpdatingEventArgs) method

Raises the RowUpdating event of a data provider.

Syntax

Visual Basic

```
Protected Overrides Sub OnRowUpdating (ByVal value As
RowUpdatingEventArgs)
```

C#

```
protected override void OnRowUpdating (RowUpdatingEventArgs value)
```

Parameters

value

A System.Data.Common.RowUpdatingEventArgs that contains the event data.

4.1.4.10.12 TerminateBatching() method

Ends batching for the HanaDataAdapter object.

Syntax

Visual Basic

```
Protected Overrides Sub TerminateBatching ()
```

C#

```
protected override void TerminateBatching ()
```

Related Information

HanaDataAdapter class [page 249]

4.1.4.10.13 Update(DataRow[], DataTableMapping) method

Updates the tables in a database with the changes made to the DataSet.

Syntax

Visual Basic

```
Protected Overrides Function Update (
    ByVal dataRows As DataRow(),
    ByVal tableMapping As DataTableMapping
) As Integer
```

C#

```
protected override int Update (
    DataRow[] dataRows,
    DataTableMapping tableMapping
)
```

Parameters

dataRows

An array of System.Data.DataRow to update from.

tableMapping

The System.Data.IDataAdapter.TableMappings collection to use.

Returns

The number of rows successfully updated from the System.Data.DataRow array.

Remarks

The Update is carried out using the InsertCommand, UpdateCommand, and DeleteCommand on each row in the data set that has been inserted, updated, or deleted.

For more information, see HanaDataAdapter: Insert rows using Add and Update and HanaDataAdapter: Delete rows using Delete and Update.

Related Information

DeleteCommand property [page 267]
InsertCommand property [page 267]

UpdateCommand property [page 270]

4.1.4.10.14 DeleteCommand property

Specifies a HanaCommand object that is executed against the database when the Update method is called to delete rows in the database that correspond to deleted rows in the DataSet.

Syntax

Visual Basic

```
Public Shadows Property DeleteCommand As HanaCommand
```

C#

```
public new HanaCommand DeleteCommand {get;set;}
```

Remarks

If this property is not set and primary key information is present in the DataSet during Update, then DeleteCommand can be generated automatically by setting SelectCommand and using the HanaCommandBuilder. In that case, the HanaCommandBuilder generates any additional commands that you do not set. This generation logic requires key column information to be present in the SelectCommand.

When DeleteCommand is assigned to an existing HanaCommand object, the HanaCommand object is not cloned. The DeleteCommand maintains a reference to the existing HanaCommand.

Related Information

SelectCommand property [page 268]

4.1.4.10.15 InsertCommand property

Specifies a HanaCommand that is executed against the database when the Update method is called that adds rows to the database to correspond to rows that were inserted in the DataSet.

Syntax

Visual Basic

```
Public Shadows Property InsertCommand As HanaCommand
```

C#

```
public new HanaCommand InsertCommand {get;set;}
```

Remarks

The HanaCommandBuilder does not require key columns to generate InsertCommand.

When InsertCommand is assigned to an existing HanaCommand object, the HanaCommand is not cloned. The InsertCommand maintains a reference to the existing HanaCommand.

If this command returns rows, then the rows may be added to the DataSet depending on how you set the UpdatedRowSource property of the HanaCommand object.

4.1.4.10.16 SelectCommand property

Specifies a HanaCommand that is used during Fill or FillSchema to obtain a result set from the database for copying into a DataSet.

Syntax

Visual Basic

```
Public Shadows Property SelectCommand As HanaCommand
```

C#

```
public new HanaCommand SelectCommand {get;set;}
```

Remarks

When SelectCommand is assigned to a previously created HanaCommand, the HanaCommand is not cloned. The SelectCommand maintains a reference to the previously created HanaCommand object.

If the SelectCommand does not return any rows, then no tables are added to the DataSet, and no exception is raised.

The SELECT statement can also be specified in the HanaDataAdapter constructor.

4.1.4.10.17 TableMappings property

Specifies a collection that provides the master mapping between a source table and a DataTable.

Syntax

Visual Basic

```
Public ReadOnly Shadows Property TableMappings As
DataTableMappingCollection
```

C#

```
public new DataTableMappingCollection TableMappings {get;}
```

Remarks

The default value is an empty collection.

When reconciling changes, the HanaDataAdapter uses the DataTableMappingCollection collection to associate the column names used by the data source with the column names used by the DataSet.

4.1.4.10.18 UpdateBatchSize property

Gets or sets the number of rows that are processed in each round-trip to the SAP HANA database server.

Syntax

Visual Basic

```
Public Overrides Property UpdateBatchSize As Integer
```

C#

```
public override int UpdateBatchSize {get;set;}
```

Remarks

The default value is 1.

Setting the value to something greater than 1 causes HanaDataAdapter.Update to execute all the insert statements in batches. The deletions and updates are executed sequentially as before, but insertions are executed afterward in batches of size equal to the value of UpdateBatchSize. Setting the value to 0 causes Update to send the insert statements in a single batch.

Setting the value to something greater than 1 causes HanaDataAdapter.Fill to execute all the insert statements in batches. The deletions and updates are executed sequentially as before, but insertions are executed afterward in batches of size equal to the value of UpdateBatchSize.

Setting the value to 0 causes Fill to send the insert statements in a single batch.

Setting it less than 0 is an error.

If UpdateBatchSize is set to something other than one, and the InsertCommand property is set to something that is not an INSERT statement, then an exception is thrown when calling Fill.

This behavior is different from SqlDataAdapter. It batches all types of commands.

4.1.4.10.19 UpdateCommand property

Specifies a HanaCommand that is executed against the database when the Update method is called to update rows in the database that correspond to updated rows in the DataSet.

Syntax

Visual Basic

```
Public Shadows Property UpdateCommand As HanaCommand
```

C#

```
public new HanaCommand UpdateCommand {get;set;}
```

Remarks

During Update, if this property is not set and primary key information is present in the SelectCommand, then the UpdateCommand can be generated automatically if you set the SelectCommand property and use the HanaCommandBuilder. Then, any additional commands that you do not set are generated by the HanaCommandBuilder. This generation logic requires key column information to be present in the SelectCommand.

When UpdateCommand is assigned to a previously created HanaCommand, the HanaCommand is not cloned. The UpdateCommand maintains a reference to the previously created HanaCommand object.

If execution of this command returns rows, then these rows can be merged with the DataSet depending on how you set the UpdatedRowSource property of the HanaCommand object.

4.1.4.10.20 RowUpdated event

Occurs during an update after a command is executed against the data source.

Syntax

Visual Basic

```
Public Event RowUpdated  As HanaRowUpdatedEventHandler
```

C#

```
public HanaRowUpdatedEventHandler RowUpdated;
```

Remarks

When an attempt to update is made, the event fires.

The event handler receives an argument of type HanaRowUpdatedEventArgs containing data related to this event.

For more information, see the .NET Framework documentation for OleDbDataAdapter.RowUpdated Event.

4.1.4.10.21 RowUpdating event

Occurs during an update before a command is executed against the data source.

Syntax

Visual Basic

```
Public Event RowUpdating  As HanaRowUpdatingEventHandler
```

C#

```
public HanaRowUpdatingEventHandler RowUpdating;
```

Remarks

When an attempt to update is made, the event fires.

The event handler receives an argument of type HanaRowUpdatingEventArgs containing data related to this event.

For more information, see the .NET Framework documentation for OleDbDataAdapter.RowUpdating Event.

4.1.4.11 HanaDataReader class

A read-only, forward-only result set from a query or stored procedure.

Syntax

Visual Basic

```
Public NotInheritable Class HanaDataReader
Inherits System.Data.Common.DbDataReader Implements
System.ComponentModel.IListSource
```

C#

```
public sealed class HanaDataReader :
System.Data.Common.DbDataReader, System.ComponentModel.IListSource
```

Members

All members of HanaDataReader, including inherited members.

Methods

Modifier and Type	Method	Description
public override void	Close() [page 274]	Closes the HanaDataReader.
public override bool	GetBoolean(int) [page 275]	Returns the value of the specified column as a Boolean.
public override byte	GetByte(int) [page 276]	Returns the value of the specified column as a Byte.
public override unsafe long	GetBytes(int, long, byte[], int, int) [page 276]	Reads a stream of bytes from the specified column offset into the buffer as an array, starting at the given buffer offset.
public override char	GetChar(int) [page 278]	Returns the value of the specified column as a character.
public override unsafe long	GetChars(int, long, char[], int, int) [page 279]	Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.
public new IDataReader	GetData(int) [page 280]	This method is not supported.
public override string	GetDataTypeName(int) [page 280]	Returns the name of the source data type.
public override DateTime	GetDateTime(int) [page 281]	Returns the value of the specified column as a DateTime object.
public DateTimeOffset	GetDateTimeOffset(int) [page 282]	Returns the value of the specified column as a DateTimeOffset object.
public override decimal	GetDecimal(int) [page 283]	Returns the value of the specified column as a Decimal object.
public override double	GetDouble(int) [page 284]	Returns the value of the specified column as a double-precision floating-point number.
public override IEnumerator	GetEnumerator() [page 285]	Returns a System.Collections.IEnumerator that iterates through the HanaDataReader object.
public override Type	GetFieldType(int) [page 285]	Returns the Type that is the data type of the object.
public override float	GetFloat(int) [page 286]	Returns the value of the specified column as a single-precision floating-point number.
public override Guid	GetGuid(int) [page 287]	Returns the value of the specified column as a global unique identifier (GUID).

Modifier and Type	Method	Description
public HanaDecimal	GetHanaDecimal(int) [page 288]	Returns the value of the specified column as a HanaDecimal object.
public override short	GetInt16(int) [page 289]	Returns the value of the specified column as a 16-bit signed integer.
public override int	GetInt32(int) [page 289]	Returns the value of the specified column as a 32-bit signed integer.
public override long	GetInt64(int) [page 290]	Returns the value of the specified column as a 64-bit signed integer.
public override string	GetName(int) [page 291]	Returns the name of the specified column.
public override int	GetOrdinal(string) [page 291]	Returns the column ordinal, given the column name.
public override unsafe DataTable	GetSchemaTable() [page 292]	Returns a DataTable that describes the column metadata of the HanaDataReader.
public override string	GetString(int) [page 294]	Returns the value of the specified column as a string.
public TimeSpan	GetTimeSpan(int) [page 295]	Returns the value of the specified column as a TimeSpan object.
public override object	GetValue(int) [page 296]	Returns the value of the specified column as an Object.
public override unsafe int	GetValues(object[]) [page 297]	Gets all the columns in the current row.
public override bool	IsDBNull(int) [page 298]	Returns a value indicating whether the column contains NULL values.
public void	myDispose() [page 298]	Frees the resources associated with the object.
public override bool	NextResult() [page 299]	Advances the HanaDataReader to the next result set when processing queries that return multiple result sets.
public override unsafe bool	Read() [page 299]	Reads the next row of the result set and moves the HanaDataReader to that row.

Properties

Modifier and Type	Property	Description
public override int	Depth [page 300]	Gets a value indicating the depth of nesting for the current row.
public override int	FieldCount [page 300]	Gets the number of columns in the result set.
public override bool	HasRows [page 301]	Gets a value that indicates whether the HanaDataReader contains one or more rows.

Modifier and Type	Property	Description
public override bool	IsClosed [page 301]	Gets a values that indicates whether the HanaDataReader is closed.
public override int	RecordsAffected [page 302]	The number of rows changed, inserted, or deleted by the execution of the SQL statement.
public override object	this [page 302]	Returns the value of a column in its native format.

Remarks

There is no constructor for HanaDataReader. To get a HanaDataReader object, execute a HanaCommand:

```
HanaCommand cmd = new HanaCommand(
   "SELECT EmployeeID FROM Employees", conn );
HanaDataReader reader = cmd.ExecuteReader();
```

You can only move forward through a HanaDataReader. If you need a more flexible object to manipulate results, then use a HanaDataAdapter.

The HanaDataReader retrieves rows as needed, whereas the HanaDataAdapter must retrieve all rows of a result set before you can carry out any action on the object. For large result sets, this difference gives the HanaDataReader a much faster response time.

Implements: IDataReader, IDisposable, IDataRecord, IListSource

For more information, see Data access and manipulation.

Related Information

ExecuteReader() method [page 169]

4.1.4.11.1 Close() method

Closes the HanaDataReader.

Syntax

Visual Basic

```
Public Overrides Sub Close ()
```

C#

```
public override void Close ()
```

Remarks

Explicitly call the Close method when you are finished using the HanaDataReader.

When running in autocommit mode, a COMMIT is issued as a side effect of closing the HanaDataReader.

4.1.4.11.2 GetBoolean(int) method

Returns the value of the specified column as a Boolean.

Syntax

Visual Basic

```
Public Overrides Function GetBoolean (ByVal ordinal As Integer) As
Boolean
```

C#

```
public override bool GetBoolean (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a Boolean.

Related Information

GetOrdinal(string) method [page 291]
GetFieldType(int) method [page 285]

4.1.4.11.3 GetByte(int) method

Returns the value of the specified column as a Byte.

Syntax

Visual Basic

```
Public Overrides Function GetByte (ByVal ordinal As Integer) As
Byte
```

C#

```
public override byte GetByte (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a byte.

4.1.4.11.4 GetBytes(int, long, byte[], int, int) method

Reads a stream of bytes from the specified column offset into the buffer as an array, starting at the given buffer offset.

Syntax

Visual Basic

```
Public Overrides Function GetBytes (
    ByVal ordinal As Integer,
    ByVal dataIndex As Long,
    ByVal buffer As Byte(),
    ByVal bufferIndex As Integer,
    ByVal length As Integer
```

```
) As Long
```

C#

```
public override unsafe long GetBytes (
    int ordinal,
    long dataIndex,
    byte[] buffer,
    int bufferIndex,
    int length
)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

dataIndex

The index within the column value from which to read bytes.

buffer

An array in which to store the data.

bufferIndex

The index in the array to start copying data.

length

The maximum length to copy into the specified buffer.

Returns

The number of bytes read.

Remarks

GetBytes returns the number of available bytes in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if GetBytes has already been used to obtain bytes from the field. This may be the case, for example, when the HanaDataReader is reading a large data structure into a buffer.

If you pass a buffer that is a null reference (Nothing in Visual Basic), then GetBytes returns the length of the field in bytes.

No conversions are performed, so the data that is being retrieved must already be a byte array.

4.1.4.11.5 GetChar(int) method

Returns the value of the specified column as a character.

Syntax

Visual Basic

```
Public Overrides Function GetChar (ByVal ordinal As Integer) As
Char
```

C#

```
public override char GetChar (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a character.

Call the HanaDataReader.IsDBNull method to check for null values before calling this method.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.6 GetChars(int, long, char[], int, int) method

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

Syntax

Visual Basic

```
Public Overrides Function GetChars (
    ByVal ordinal As Integer,
    ByVal dataIndex As Long,
    ByVal buffer As Char(),
    ByVal bufferIndex As Integer,
    ByVal length As Integer
) As Long
```

C#

```
public override unsafe long GetChars (
    int ordinal,
    long dataIndex,
    char[] buffer,
    int bufferIndex,
    int length
)
```

Parameters

ordinal

The zero-based column ordinal.

dataIndex

The index within the row from which to begin the read operation.

buffer

The buffer into which to copy data.

bufferIndex

The index for buffer to begin the read operation.

length

The number of characters to read.

Returns

The actual number of characters read.

Remarks

GetChars returns the number of available characters in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if GetChars has already been used to obtain characters from the field. This may be the case, for example, when the HanaDataReader is reading a large data structure into a buffer.

If you pass a buffer that is a null reference (Nothing in Visual Basic), then GetChars returns the length of the field in characters.

No conversions are performed, so the data that is being retrieved must already be a character array.

For information about handling BLOBs, see BLOBs.

4.1.4.11.7 GetData(int) method

This method is not supported.

Syntax

Visual Basic

```
Public Shadows Function GetData (ByVal i As Integer) As IDataReader
```

C#

```
public new IDataReader GetData (int i)
```

Remarks

When called, it throws an InvalidOperationException.

4.1.4.11.8 GetDataTypeName(int) method

Returns the name of the source data type.

Syntax

Visual Basic

```
Public Overrides Function GetDataTypeName (ByVal index As Integer)
As String
```

C#

```
public override string GetDataTypeName (int index)
```

Parameters

index

The zero-based column ordinal.

Returns

The name of the back-end data type.

4.1.4.11.9 GetDateTime(int) method

Returns the value of the specified column as a DateTime object.

Syntax

Visual Basic

```
Public Overrides Function GetDateTime (ByVal ordinal As Integer)
As Date
```

C#

```
public override DateTime GetDateTime (int ordinal)
```

Parameters

ordinal

The zero-based column ordinal.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a DateTime object.

Call the HanaDataReader.IsDBNull method to check for null values before calling this method.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.10 GetDateTimeOffset(int) method

Returns the value of the specified column as a DateTimeOffset object.

Syntax

Visual Basic

```
Public Function GetDateTimeOffset (ByVal ordinal As Integer) As
DateTimeOffset
```

C#

```
public DateTimeOffset GetDateTimeOffset (int ordinal)
```

Parameters

ordinal

The zero-based column ordinal.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a DateTimeOffset object.

Call the HanaDataReader.IsDBNull method to check for null values before calling this method.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.11 GetDecimal(int) method

Returns the value of the specified column as a Decimal object.

Syntax

Visual Basic

```
Public Overrides Function GetDecimal (ByVal ordinal As Integer) As
Decimal
```

C#

```
public override decimal GetDecimal (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a Decimal object.

Call the HanaDataReader.IsDBNull method to check for null values before calling this method.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.12 GetDouble(int) method

Returns the value of the specified column as a double-precision floating-point number.

Syntax

Visual Basic

```
Public Overrides Function GetDouble (ByVal ordinal As Integer) As
Double
```

C#

```
public override double GetDouble (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a double-precision floating-point number.

Call the HanaDataReader.IsDBNull method to check for null values before calling this method.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.13 GetEnumerator() method

Returns a System.Collections.IEnumerator that iterates through the HanaDataReader object.

Syntax

Visual Basic

```
Public Overrides Function GetEnumerator () As
System.Collections.IEnumerator
```

C#

```
public override IEnumerator GetEnumerator ()
```

Returns

A System.Collections.IEnumerator for the HanaDataReader object.

Related Information

HanaDataReader class [page 271]

4.1.4.11.14 GetFieldType(int) method

Returns the Type that is the data type of the object.

Syntax

Visual Basic

```
Public Overrides Function GetFieldType (ByVal index As Integer) As
Type
```

C#

```
public override Type GetFieldType (int index)
```

Parameters

index

The zero-based column ordinal.

Returns

The type that is the data type of the object.

4.1.4.11.15 GetFloat(int) method

Returns the value of the specified column as a single-precision floating-point number.

Syntax

Visual Basic

```
Public Overrides Function GetFloat (ByVal ordinal As Integer) As
Single
```

C#

```
public override float GetFloat (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a single-precision floating-point number.

Call the HanaDataReader.IsDBNull method to check for null values before calling this method.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.16 GetGuid(int) method

Returns the value of the specified column as a global unique identifier (GUID).

Syntax

Visual Basic

```
Public Overrides Function GetGuid (ByVal ordinal As Integer) As
Guid
```

C#

```
public override Guid GetGuid (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

The data that is being retrieved must already be a globally-unique identifier or binary(16).

Call the HanaDataReader.IsDBNull method to check for null values before calling this method.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.17 GetHanaDecimal(int) method

Returns the value of the specified column as a HanaDecimal object.

Syntax

Visual Basic

```
Public Function GetHanaDecimal (ByVal ordinal As Integer) As
HanaDecimal
```

C#

```
public HanaDecimal GetHanaDecimal (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a decimal value.

Call the HanaDataReader.IsDBNull method to check for null values before calling this method.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.18 GetInt16(int) method

Returns the value of the specified column as a 16-bit signed integer.

Syntax

Visual Basic

```
Public Overrides Function GetInt16 (ByVal ordinal As Integer) As
Short
```

C#

```
public override short GetInt16 (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a 16-bit signed integer.

4.1.4.11.19 GetInt32(int) method

Returns the value of the specified column as a 32-bit signed integer.

Syntax

Visual Basic

```
Public Overrides Function GetInt32 (ByVal ordinal As Integer) As
Integer
```

C#

```
public override int GetInt32 (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a 32-bit signed integer.

4.1.4.11.20 GetInt64(int) method

Returns the value of the specified column as a 64-bit signed integer.

Syntax

Visual Basic

```
Public Overrides Function GetInt64 (ByVal ordinal As Integer) As
Long
```

C#

```
public override long GetInt64 (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a 64-bit signed integer.

4.1.4.11.21 GetName(int) method

Returns the name of the specified column.

Syntax

Visual Basic

```
Public Overrides Function GetName (ByVal index As Integer) As
String
```

C#

```
public override string GetName (int index)
```

Parameters

index

The zero-based index of the column.

Returns

The name of the specified column.

4.1.4.11.22 GetOrdinal(string) method

Returns the column ordinal, given the column name.

Syntax

Visual Basic

```
Public Overrides Function GetOrdinal (ByVal name As String) As
Integer
```

C#

```
public override int GetOrdinal (string name)
```

Parameters

name

The column name.

Returns

The zero-based column ordinal.

Remarks

GetOrdinal performs a case-sensitive lookup first. If it fails, then a second case-insensitive search is made.

GetOrdinal is Japanese kana-width insensitive.

Because ordinal-based lookups are more efficient than named lookups, it is inefficient to call GetOrdinal within a loop. Save time by calling GetOrdinal once and assigning the results to an integer variable for use within the loop.

4.1.4.11.23 GetSchemaTable() method

Returns a DataTable that describes the column metadata of the HanaDataReader.

Syntax

Visual Basic

```
Public Overrides Function GetSchemaTable () As DataTable
```

C#

```
public override unsafe DataTable GetSchemaTable ()
```

Returns

A DataTable that describes the column metadata.

Remarks

This method returns metadata about each column in the following order:

DataTable column	Description
ColumnName	The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, then the alias is returned. In result sets, not all columns have names and not all column names are unique.
ColumnOrdinal	The ID of the column. The value is in the range [0, FieldCount -1].
ColumnSize	For sized columns, the maximum length of a value in the column. For other columns, this is the size in bytes of the data type.
NumericPrecision	The precision of a numeric column or DBNull if the column is not numeric.
NumericScale	The scale of a numeric column or DBNull if the column is not numeric.
IsUnique	True if the column is a non-computed unique column in the table (BaseTableName) it is taken from.
IsKey	True if the column is one of a set of columns in the result set that taken together from a unique key for the result set. The set of columns with IsKey set to true does not need to be the minimal set that uniquely identifies a row in the result set.
BaseServerName	The name of the SAP HANA database server used by the HanaDataReader.
BaseCatalogName	The name of the catalog in the database that contains the column. This value is always DBNull.
BaseColumnName	The original name of the column in the table BaseTableName of the database or DBNull if the column is computed or if this information cannot be determined.
BaseSchemaName	The name of the schema in the database that contains the column.
BaseTableName	The name of the table in the database that contains the column, or DBNull if column is computed or if this information cannot be determined.
DataType	The .NET data type that is most appropriate for this type of column.
AllowDBNull	True if the column is nullable; false if the column is not nullable or if this information cannot be determined.
ProviderType	The type of the column.
IsAliased	True if the column name is an alias; false if it is not an alias.
IsExpression	True if the column is an expression; false if it is a column value.

DataTable column	Description
IsIdentity	True if the column is an identity column; false if it is not an identity column.
IsAutoIncrement	True if the column is an autoincrement or global autoincrement column; false otherwise (or if this information cannot be determined).
IsRowVersion	True if the column contains a persistent row identifier that cannot be written to, and has no meaningful value except to identify the row.
IsHidden	True if the column is hidden; false otherwise.
IsLong	True if the column is a BLOB, CLOB, NCLOB, or a TEXT column; false otherwise.
IsReadOnly	True if the column is read-only; false if the column is modifiable or if its access cannot be determined.

For more information about these columns, see the .NET Framework documentation for SqlDataReader.GetSchemaTable.

For more information, see HanaCommand: Fetch result set schema using GetSchemaTable.

4.1.4.11.24 GetString(int) method

Returns the value of the specified column as a string.

Syntax

Visual Basic

```
Public Overrides Function GetString (ByVal ordinal As Integer) As String
```

C#

```
public override string GetString (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

No conversions are performed, so the data that is being retrieved must already be a string.

Call the HanaDataReader.IsDBNull method to check for NULL values before calling this method.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.25 GetTimeSpan(int) method

Returns the value of the specified column as a TimeSpan object.

Syntax

Visual Basic

```
Public Function GetTimeSpan (ByVal ordinal As Integer) As TimeSpan
```

C#

```
public TimeSpan GetTimeSpan (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column.

Remarks

The column must be an SAP HANA TIME data type. The data is converted to TimeSpan. The Days property of TimeSpan is always set to 0.

Call HanaDataReader.IsDBNull method to check for NULL values before calling this method.

For more information, see Time values.

Related Information

IsDBNull(int) method [page 298]

4.1.4.11.26 GetValue(int) method

Returns the value of the specified column as an Object.

Syntax

Visual Basic

```
Public Overrides Function GetValue (ByVal ordinal As Integer) As
Object
```

C#

```
public override object GetValue (int ordinal)
```

Parameters

ordinal

An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Returns

The value of the specified column as an object.

Remarks

This method returns DBNull for NULL database columns.

4.1.4.11.27 GetValues(object[]) method

Gets all the columns in the current row.

Syntax

Visual Basic

```
Public Overrides Function GetValues (ByVal values As Object()) As
Integer
```

C#

```
public override unsafe int GetValues (object[] values)
```

Parameters

values

An array of objects that holds an entire row of the result set.

Returns

The number of objects in the array.

Remarks

For most applications, the GetValues method provides an efficient means for retrieving all columns, rather than retrieving each column individually.

You can pass an Object array that contains fewer than the number of columns contained in the resulting row. Only the amount of data the Object array holds is copied to the array. Also, you can pass an Object array whose length is more than the number of columns contained in the resulting row.

This method returns DBNull for NULL database columns.

4.1.4.11.28 IsDBNull(int) method

Returns a value indicating whether the column contains NULL values.

Syntax

Visual Basic

```
Public Overrides Function IsDBNull (ByVal ordinal As Integer) As
Boolean
```

C#

```
public override bool IsDBNull (int ordinal)
```

Parameters

ordinal

The zero-based column ordinal.

Returns

True if the specified column value is equivalent to DBNull; false otherwise.

Remarks

Call this method to check for NULL column values before calling the typed get methods (for example, GetByte, GetChar, and so on) to avoid raising an exception.

4.1.4.11.29 myDispose() method

Frees the resources associated with the object.

Syntax

Visual Basic

```
Public Sub myDispose ()
```

C#

```
public void myDispose ()
```

4.1.4.11.30 NextResult() method

Advances the HanaDataReader to the next result set when processing queries that return multiple result sets.

Syntax

Visual Basic

```
Public Overrides Function NextResult () As Boolean
```

C#

```
public override bool NextResult ()
```

Returns

True if there are more result sets; false otherwise.

Remarks

Used to process multiple result sets, which can be generated by executing batch SQL statements or stored procedures.

By default, the data reader is positioned on the first result set.

4.1.4.11.31 Read() method

Reads the next row of the result set and moves the HanaDataReader to that row.

Syntax

Visual Basic

```
Public Overrides Function Read () As Boolean
```

C#

```
public override unsafe bool Read ()
```

Returns

True if there are more rows; false otherwise.

Remarks

The default position of the HanaDataReader is prior to the first record. Call Read to begin accessing any data.

Example

The following code fills a listbox with the values in a single column of results.

```
while( reader.Read() ) {
   listResults.Items.Add( reader.GetValue( 0 ).ToString() );
}
listResults.EndUpdate();
reader.Close();
```

4.1.4.11.32 Depth property

Gets a value indicating the depth of nesting for the current row.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property Depth As Integer
```

C#

```
public override int Depth {get;}
```

Remarks

The outermost table has a depth of zero.

The depth of nesting for the current row.

4.1.4.11.33 FieldCount property

Gets the number of columns in the result set.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property FieldCount As Integer
```

C#

```
public override int FieldCount {get;}
```

Remarks

The number of columns in the current record.

4.1.4.11.34 HasRows property

Gets a value that indicates whether the HanaDataReader contains one or more rows.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property HasRows As Boolean
```

C#

```
public override bool HasRows {get;}
```

Remarks

True if the HanaDataReader contains one or more rows; false otherwise.

4.1.4.11.35 IsClosed property

Gets a values that indicates whether the HanaDataReader is closed.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property IsClosed As Boolean
```

C#

```
public override bool IsClosed {get;}
```

Remarks

True if the HanaDataReader is closed; false otherwise.

IsClosed and RecordsAffected are the only properties that you can use after the HanaDataReader is closed.

4.1.4.11.36 RecordsAffected property

The number of rows changed, inserted, or deleted by the execution of the SQL statement.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property RecordsAffected As Integer
```

C#

```
public override int RecordsAffected {get;}
```

Remarks

The number of rows changed, inserted, or deleted. This value is 0 if no rows were affected or the statement failed, or -1 for SELECT statements.

The number of rows changed, inserted, or deleted. This value is 0 if no rows were affected or the statement failed, and -1 for SELECT statements.

The value of this property is cumulative. For example, if two records are inserted in batch mode, then the value of RecordsAffected is 2.

IsClosed and RecordsAffected are the only properties that you can use after the HanaDataReader is closed.

4.1.4.11.37 this property

Returns the value of a column in its native format.

Overload list

Modifier and Type	Overload name	Description
public override object	this[int index] [page 303]	Returns the value of a column in its native format.

Modifier and Type	Overload name	Description
public override object	this[string name] [page 303]	Returns the value of a column in its native format.

4.1.4.11.37.1 this[int index] property

Returns the value of a column in its native format.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property Item (ByVal index As Integer) As Object
```

C#

```
public override object this[int index] {get;}
```

Remarks

In C#, this property is the indexer for the HanaDataReader class.

4.1.4.11.37.2 this[string name] property

Returns the value of a column in its native format.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property Item (ByVal name As String) As Object
```

C#

```
public override object this[string name] {get;}
```

Remarks

In C#, this property is the indexer for the HanaDataReader class.

4.1.4.12 HanaDataSourceEnumerator class

Provides a mechanism for enumerating all available instances of SAP HANA database servers within the local network.

Syntax

Visual Basic

```
Public NotInheritable Class HanaDataSourceEnumerator Inherits
System.Data.Common.DbDataSourceEnumerator
```

C#

```
public sealed class HanaDataSourceEnumerator :
System.Data.Common.DbDataSourceEnumerator
```

Members

All members of HanaDataSourceEnumerator, including inherited members.

Methods

Modifier and Type	Method	Description
public unsafe override DataTable	GetDataSources() [page 305]	Retrieves a DataTable containing information about all visible SAP HANA database servers.

Properties

Modifier and Type	Property	Description
public HanaDataSourceEnumerator	Instance [page 305]	Gets an instance of HanaDataSourceEnumerator, which can be used to retrieve information about all visible SAP HANA database servers.

Remarks

There is no constructor for HanaDataSourceEnumerator.

4.1.4.12.1 GetDataSources() method

Retrieves a DataTable containing information about all visible SAP HANA database servers.

Syntax

Visual Basic

```
Public Overrides Function GetDataSources () As DataTable
```

C#

```
public unsafe override DataTable GetDataSources ()
```

Remarks

The returned table has four columns: ServerName, IPAddress, PortNumber, and DataBaseNames. There is a row in the table for each available database server.

Example

The following code fills a DataTable with information for each database server that is available.

```
DataTable servers = HanaDataSourceEnumerator.Instance.GetDataSources();
```

4.1.4.12.2 Instance property

Gets an instance of HanaDataSourceEnumerator, which can be used to retrieve information about all visible SAP HANA database servers.

Syntax

Visual Basic

```
Public Shared ReadOnly Property Instance As
HanaDataSourceEnumerator
```

C#

```
public HanaDataSourceEnumerator Instance {get;}
```

4.1.4.13 HanaDefault class

Represents a parameter with a default value.

Syntax

Visual Basic

```
Public NotInheritable Class HanaDefault
```

C#

```
public sealed class HanaDefault
```

Members

All members of HanaDefault, including inherited members.

Variables

Modifier and Type	Variable	Description
public static readonly HanaDefault	Value	Gets the value for a default parameter. This field is read-only and static.

Remarks

There is no constructor for HanaDefault.

```
HanaParameter parm = new HanaParameter();
parm.Value = HanaDefault.Value;
```

4.1.4.14 HanaError class

Collects information relevant to a warning or error returned by the data source.

Syntax

Visual Basic

```
Public NotInheritable Class HanaError
```

C#

```
public sealed class HanaError
```

Members

All members of HanaError, including inherited members.

Methods

Modifier and Type	Method	Description
public override string	ToString() [page 307]	The complete text of the error message.

Properties

Modifier and Type	Property	Description
public string	Message [page 308]	Returns a short description of the error.
public int	NativeError [page 308]	Returns database-specific error information.
public string	Source [page 308]	Returns the name of the provider that generated the error.
public string	SqlState [page 309]	The five-character SQLSTATE following the ANSI SQL standard.

Remarks

There is no constructor for HanaError.

For information about error handling, see Error handling.

4.1.4.14.1 ToString() method

The complete text of the error message.

Syntax

Visual Basic

```
Public Overrides Function ToString () As String
```

C#

```
public override string ToString ()
```

Example

The return value is a string is in the form *HanaError:*, followed by the message. For example:

```
HanaError:UserId or Password not valid.
```

4.1.4.14.2 Message property

Returns a short description of the error.

Syntax

Visual Basic

```
Public ReadOnly Property Message As String
```

C#

```
public string Message {get;}
```

4.1.4.14.3 NativeError property

Returns database-specific error information.

Syntax

Visual Basic

```
Public ReadOnly Property NativeError As Integer
```

C#

```
public int NativeError {get;}
```

4.1.4.14.4 Source property

Returns the name of the provider that generated the error.

Syntax

Visual Basic

```
Public ReadOnly Property Source As String
```

C#

```
public string Source {get;}
```

4.1.4.14.5 SqlState property

The five-character SQLSTATE following the ANSI SQL standard.

Syntax

Visual Basic

```
Public ReadOnly Property SqlState As String
```

C#

```
public string SqlState {get;}
```

4.1.4.15 HanaErrorCollection class

Collects all errors generated by the data provider.

Syntax

Visual Basic

```
Public NotInheritable Class HanaErrorCollection Implements
System.Collections.ICollection, System.Collections.IEnumerable
```

C#

```
public sealed class HanaErrorCollection :
System.Collections.ICollection, System.Collections.IEnumerable
```

Members

All members of HanaErrorCollection, including inherited members.

Methods

Modifier and Type	Method	Description
public void	CopyTo(Array, int) [page 310]	Copies the elements of the HanaError-Collection into an array, starting at the given index within the array.
public IEnumerator	GetEnumerator() [page 311]	Returns an enumerator that iterates through the HanaErrorCollection.

Properties

Modifier and Type	Property	Description
public int	Count [page 311]	Returns the number of errors in the collection.

Modifier and Type	Property	Description
public HanaError	this[int index] [page 311]	Returns the error at the specified index.

Remarks

There is no constructor for HanaErrorCollection. Typically, a HanaErrorCollection is obtained from the HanaException.Errors property.

Implements: ICollection, IEnumerable

Related Information

Errors property [page 314]

4.1.4.15.1 CopyTo(Array, int) method

Copies the elements of the HanaErrorCollection into an array, starting at the given index within the array.

Syntax

Visual Basic

```
Public Sub CopyTo (
    ByVal array As Array,
    ByVal index As Integer
)
```

C#

```
public void CopyTo (
    Array array,
    int index
)
```

Parameters

array

The array into which to copy the elements.

index

The starting index of the array.

4.1.4.15.2 GetEnumerator() method

Returns an enumerator that iterates through the HanaErrorCollection.

Syntax

Visual Basic

```
Public Function GetEnumerator () As System.Collections.IEnumerator
```

C#

```
public IEnumerator GetEnumerator ()
```

Returns

A System.Collections.IEnumerator for the HanaErrorCollection.

4.1.4.15.3 Count property

Returns the number of errors in the collection.

Syntax

Visual Basic

```
Public ReadOnly Property Count As Integer
```

C#

```
public int Count {get;}
```

4.1.4.15.4 this[int index] property

Returns the error at the specified index.

Syntax

Visual Basic

```
Public ReadOnly Property Item (ByVal index As Integer) As HanaError
```

C#

```
public HanaError this[int index] {get;}
```

Remarks

A HanaError object that contains the error at the specified index.

Related Information

HanaError class [page 306]

4.1.4.16 HanaException class

The exception that is thrown when the SAP HANA database server returns a warning or error.

Syntax

Visual Basic

```
Public Class HanaException Inherits System.Exception
```

C#

```
public class HanaException : System.Exception
```

Members

All members of HanaException, including inherited members.

Methods

Modifier and Type	Method	Description
public override void	GetObjectData(SerializationInfo, StreamingContext) [page 313]	Sets the SerializationInfo with information about the exception.

Properties

Modifier and Type	Property	Description
public HanaErrorCollection	Errors [page 314]	Returns a collection of one or more HanaError objects.
public override string	Message [page 314]	Returns the text describing the error.
public int	NativeError [page 315]	Returns database-specific error information.
public override string	Source [page 315]	Returns the name of the provider that generated the error.

Remarks

There is no constructor for HanaException. Typically, a HanaException object is declared in a catch. For example:

```
...
catch( HanaException ex )
{
   MessageBox.Show( ex.Errors[0].Message, "Error" );
}
```

For information about error handling, see Error handling.

4.1.4.16.1 GetObjectData(SerializationInfo, StreamingContext) method

Sets the SerializationInfo with information about the exception.

Syntax

Visual Basic

```
Public Overrides Sub GetObjectData (
    ByVal info As SerializationInfo,
    ByVal context As StreamingContext
)
```

C#

```
public override void GetObjectData (
    SerializationInfo info,
    StreamingContext context
)
```

Parameters

info

The SerializationInfo that holds the serialized object data about the exception being thrown.

context

The StreamingContext that contains contextual information about the source or destination.

Remarks

Overrides Exception.GetObjectData.

4.1.4.16.2 Errors property

Returns a collection of one or more HanaError objects.

Syntax

Visual Basic

```
Public ReadOnly Property Errors As HanaErrorCollection
```

C#

```
public HanaErrorCollection Errors {get;}
```

Remarks

The HanaErrorCollection object always contains at least one instance of the HanaError object.

Related Information

HanaErrorCollection class [page 309]
HanaError class [page 306]

4.1.4.16.3 Message property

Returns the text describing the error.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property Message As String
```

C#

```
public override string Message {get;}
```

Remarks

This method returns a single string that contains a concatenation of all of the Message properties of all of the HanaError objects in the Errors collection. Each message, except the last one, is followed by a carriage return.

Related Information

HanaError class [page 306]

4.1.4.16.4 NativeError property

Returns database-specific error information.

Syntax

Visual Basic

```
Public ReadOnly Property NativeError As Integer
```

C#

```
public int NativeError {get;}
```

4.1.4.16.5 Source property

Returns the name of the provider that generated the error.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property Source As String
```

C#

```
public override string Source {get;}
```

4.1.4.17 HanaFactory class

Represents a set of methods for creating instances of the Sap.Data.Hana provider's implementation of the data source classes.

Syntax

Visual Basic

```
Public NotInheritable Class HanaFactory Inherits
System.Data.Common.DbProviderFactory
```

C#

```
public sealed class HanaFactory :
System.Data.Common.DbProviderFactory
```

Members

All members of HanaFactory, including inherited members.

Variables

Modifier and Type	Variable	Description
public static readonly HanaFactory	Instance	Represents the singleton instance of the HanaFactory class. HanaFactory is a singleton class, which means only this instance of this class can exist. Normally you would not use this field directly. Instead, you get a reference to this instance of HanaFactory using System.Data.Common.DbProviderFactories.GetFactory(String). For an example, see the HanaFactory description.

Methods

Modifier and Type	Method	Description
public override DbCommand	CreateCommand() [page 317]	Returns a strongly typed System.Data.Common.DbCommand instance.
public override DbCommandBuilder	CreateCommandBuilder() [page 318]	Returns a strongly typed System.Data.Common.DbCommandBuilder instance.
public override DbConnection	CreateConnection() [page 318]	Returns a strongly typed System.Data.Common.DbConnection instance.
public override DbConnectionStringBuilder	CreateConnectionStringBuilder() [page 319]	Returns a strongly typed System.Data.Common.DbConnectionStringBuilder instance.
public override DbDataAdapter	CreateDataAdapter() [page 320]	Returns a strongly typed System.Data.Common.DbDataAdapter instance.
public override DbParameter	CreateParameter() [page 320]	Returns a strongly typed System.Data.Common.DbParameter instance.
public override CodeAccessPermission	CreatePermission(PermissionState) [page 321]	Returns a strongly-typed CodeAccessPermission instance.

Properties

Modifier and Type	Property	Description
public override bool	CanCreateDataSourceEnumerator [page 322]	Always returns false, which indicates that a HanaDataSourceEnumerator object cannot be created.

Remarks

There is no constructor for HanaFactory.

DbProviderFactories and DbProviderFactory make provider independent code easier to write. To use them with SAP HANA, specify Sap.Data.Hana as the provider invariant name passed to GetFactory. For example:

```
' Visual Basic
Dim factory As DbProviderFactory = _
 DbProviderFactories.GetFactory( "Sap.Data.Hana" )
Dim conn As DbConnection = _
 factory.CreateConnection()
// C#
DbProviderFactory factory =
        DbProviderFactories.GetFactory("Sap.Data.Hana" );
DbConnection conn = factory.CreateConnection();
```

In this example, conn is created as a HanaConnection object.

For an explanation of provider factories and generic programming in ADO.NET 2.0, see Generic Coding with the ADO.NET 2.0 Base Classes and Factories .

4.1.4.17.1 CreateCommand() method

Returns a strongly typed System.Data.Common.DbCommand instance.

Syntax

Visual Basic

```
Public Overrides Function CreateCommand () As DbCommand
```

C#

```
public override DbCommand CreateCommand ()
```

Returns

A new HanaCommand object typed as DbCommand.

Related Information

HanaCommand class [page 147]

4.1.4.17.2 CreateCommandBuilder() method

Returns a strongly typed System.Data.Common.DbCommandBuilder instance.

Syntax

Visual Basic

```
Public Overrides Function CreateCommandBuilder () As
DbCommandBuilder
```

C#

```
public override DbCommandBuilder CreateCommandBuilder ()
```

Returns

A new HanaCommandBuilder object typed as DbCommandBuilder.

Related Information

HanaCommandBuilder class [page 183]

4.1.4.17.3 CreateConnection() method

Returns a strongly typed System.Data.Common.DbConnection instance.

Syntax

Visual Basic

```
Public Overrides Function CreateConnection () As DbConnection
```

C#

```
public override DbConnection CreateConnection ()
```

Returns

A new HanaConnection object typed as DbConnection.

Related Information

HanaConnection class [page 203]

4.1.4.17.4 CreateConnectionStringBuilder() method

Returns a strongly typed System.Data.Common.DbConnectionStringBuilder instance.

Syntax

Visual Basic

```
Public Overrides Function CreateConnectionStringBuilder () As
DbConnectionStringBuilder
```

C#

```
public override DbConnectionStringBuilder
CreateConnectionStringBuilder ()
```

Returns

A new HanaConnectionStringBuilder object typed as DbConnectionStringBuilder.

Related Information

HanaConnectionStringBuilder class [page 230]

4.1.4.17.5 CreateDataAdapter() method

Returns a strongly typed System.Data.Common.DbDataAdapter instance.

Syntax

Visual Basic

```
Public Overrides Function CreateDataAdapter () As DbDataAdapter
```

C#

```
public override DbDataAdapter CreateDataAdapter ()
```

Returns

A new HanaDataAdapter object typed as DbDataAdapter.

Related Information

HanaDataAdapter class [page 249]

4.1.4.17.6 CreateParameter() method

Returns a strongly typed System.Data.Common.DbParameter instance.

Syntax

Visual Basic

```
Public Overrides Function CreateParameter () As DbParameter
```

C#

```
public override DbParameter CreateParameter ()
```

Returns

A new HanaParameter object typed as DbParameter.

Related Information

HanaParameter class [page 327]

4.1.4.17.7 CreatePermission(PermissionState) method

Returns a strongly-typed CodeAccessPermission instance. This method is not supported for .NET Core.

Syntax

Microsoft Visual Basic

```
Public Overrides Function CreatePermission (ByVal state As
PermissionState) As CodeAccessPermission
```

Microsoft C#

```
public override CodeAccessPermission CreatePermission
(PermissionState state)
```

Parameters

state

A member of the System.Security.Permissions.PermissionState enumeration.

Returns

A new HanaPermission object typed as CodeAccessPermission.

Related Information

HanaPermission class [page 368]

4.1.4.17.8 CanCreateDataSourceEnumerator property

Always returns false, which indicates that a HanaDataSourceEnumerator object cannot be created.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property CanCreateDataSourceEnumerator
As Boolean
```

C#

```
public override bool CanCreateDataSourceEnumerator {get;}
```

Returns

False.

4.1.4.18 HanaInfoMessageEventArgs class

Provides data for the InfoMessage event.

Syntax

Visual Basic

```
Public NotInheritable Class HanaInfoMessageEventArgs Inherits
System.EventArgs
```

C#

```
public sealed class HanaInfoMessageEventArgs : System.EventArgs
```

Members

All members of HanaInfoMessageEventArgs, including inherited members.

Methods

Modifier and Type	Method	Description
public override string	ToString() [page 323]	Retrieves a string representation of the InfoMessage event.

Properties

Modifier and Type	Property	Description
public HanaErrorCollection	Errors [page 323]	Returns the collection of messages sent from the data source.
public string	Message [page 324]	Returns the full text of the error sent from the data source.
public HanaMessageType	MessageType [page 324]	Returns the type of the message.
public int	NativeError [page 324]	Returns the SQLCODE returned by the SAP HANA database server.
public string	Source [page 325]	Returns the name of the data provider.

Remarks

There is no constructor for HanaInfoMessageEventArgs.

4.1.4.18.1 ToString() method

Retrieves a string representation of the InfoMessage event.

Syntax

Visual Basic

```
Public Overrides Function ToString () As String
```

C#

```
public override string ToString ()
```

Returns

A string representing the InfoMessage event.

4.1.4.18.2 Errors property

Returns the collection of messages sent from the data source.

Syntax

Visual Basic

```
Public ReadOnly Property Errors As HanaErrorCollection
```

C#

```
public HanaErrorCollection Errors {get;}
```

4.1.4.18.3 Message property

Returns the full text of the error sent from the data source.

Syntax

Visual Basic

```
Public ReadOnly Property Message As String
```

C#

```
public string Message {get;}
```

4.1.4.18.4 MessageType property

Returns the type of the message.

Syntax

Visual Basic

```
Public ReadOnly Property MessageType As HanaMessageType
```

C#

```
public HanaMessageType MessageType {get;}
```

Remarks

This can be one of: Action, Info, Status, or Warning.

4.1.4.18.5 NativeError property

Returns the SQLCODE returned by the SAP HANA database server.

Syntax

Visual Basic

```
Public ReadOnly Property NativeError As Integer
```

C#

```
public int NativeError {get;}
```

4.1.4.18.6 Source property

Returns the name of the data provider.

Syntax

Visual Basic

```
Public ReadOnly Property Source As String
```

C#

```
public string Source {get;}
```

4.1.4.19 HanaMetaDataCollectionNames class

Provides a list of constants for use with the HanaConnection.GetSchema(string) method to retrieve metadata collections.

Syntax

Visual Basic

```
Public NotInheritable Class HanaMetaDataCollectionNames
```

C#

```
public sealed class HanaMetaDataCollectionNames
```

Members

All members of HanaMetaDataCollectionNames, including inherited members.

Variables

Modifier and Type	Variable	Description
public static readonly string	Columns	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the Columns collection.

Modifier and Type	Variable	Description
public static readonly string	DataSourceInformation	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the DataSourceInformation collection.
public static readonly string	DataTypes	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the DataTypes collection.
public static readonly string	ForeignKeys	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the ForeignKeys collection.
public static readonly string	IndexColumns	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the IndexColumns collection.
public static readonly string	Indexes	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the Indexes collection.
public static readonly string	MetaDataCollections	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the MetaDataCollections collection.
public static readonly string	ProcedureParameters	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the ProcedureParameters collection.
public static readonly string	Procedures	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the Procedures collection.
public static readonly string	ReservedWords	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the ReservedWords collection.
public static readonly string	Restrictions	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the Restrictions collection.
public static readonly string	Tables	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the Tables collection.
public static readonly string	Users	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the Users collection.

Modifier and Type	Variable	Description
public static readonly string	ViewColumns	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the ViewColumns collection.
public static readonly string	Views	Provides a constant for use with the HanaConnection.GetSchema(string) method that represents the Views collection.

Remarks

This field is constant and read-only.

Related Information

GetSchema(string) method [page 218]

4.1.4.20 HanaParameter class

Represents a parameter to a HanaCommand, and optionally, its mapping to a DataSet column.

Syntax

Visual Basic

```
Public NotInheritable Class HanaParameter Inherits
System.Data.Common.DbParameter Implements System.ICloneable
```

C#

```
public sealed class HanaParameter :
System.Data.Common.DbParameter, System.ICloneable
```

Members

All members of HanaParameter, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaParameter [page 329]	Initializes a HanaParameter object with null (Nothing in Visual Basic) as its value.

Methods

Modifier and Type	Method	Description
public override void	ResetDbType() [page 334]	Resets the type (the values of DbType and HanaDbType) associated with this HanaParameter.
public override string	ToString() [page 334]	Returns a string containing the ParameterName.

Properties

Modifier and Type	Property	Description
public override DbType	DbType [page 335]	Gets and sets the DbType of the parameter.
public override ParameterDirection	Direction [page 335]	Gets and sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
public HanaDbType	HanaDbType [page 336]	The HanaDbType of the parameter.
public override bool	IsNullable [page 336]	Gets and sets a value indicating whether the parameter accepts null values.
public int	Offset [page 337]	Gets and sets the offset to the Value property.
public override string	ParameterName [page 337]	Gets and sets the name of the HanaParameter.
public byte	Precision [page 338]	Gets and sets the maximum number of digits used to represent the Value property.
public byte	Scale [page 338]	Gets and sets the number of decimal places to which Value is resolved.
public override int	Size [page 339]	Gets and sets the maximum size, in bytes, of the data within the column.
public override string	SourceColumn [page 339]	Gets and sets the name of the source column mapped to the DataSet and used for loading or returning the value.
public override bool	SourceColumnNullMapping [page 340]	Gets and sets value that indicates whether the source column is nullable.
public override DataRowVersion	SourceVersion [page 340]	Gets and sets the DataRowVersion to use when loading Value.

Modifier and Type	Property	Description
public override object	Value [page 341]	Gets and sets the value of the parameter.

Remarks

Implements: IDbDataParameter, IDataParameter, ICloneable

4.1.4.20.1 HanaParameter constructor

Initializes a HanaParameter object with null (Nothing in Visual Basic) as its value.

Overload list

Modifier and Type	Overload name	Description
public	HanaParameter() [page 330]	Initializes a HanaParameter object with null (Nothing in Visual Basic) as its value.
public	HanaParameter(string, HanaDbType) [page 330]	Initializes a HanaParameter object with the specified parameter name and data type.
public	HanaParameter(string, HanaDbType, int) [page 331]	Initializes a HanaParameter object with the specified parameter name and data type.
public	HanaParameter(string, HanaDbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object) [page 331]	Initializes a HanaParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source column, source version, and value.
public	HanaParameter(string, HanaDbType, int, string) [page 333]	Initializes a HanaParameter object with the specified parameter name, data type, and length.
public	HanaParameter(string, object) [page 333]	Initializes a HanaParameter object with the specified parameter name and value.

4.1.4.20.1.1 HanaParameter() constructor

Initializes a HanaParameter object with null (Nothing in Visual Basic) as its value.

Syntax

Visual Basic

```
Public Sub HanaParameter ()
```

C#

```
public HanaParameter ()
```

4.1.4.20.1.2 HanaParameter(string, HanaDbType) constructor

Initializes a HanaParameter object with the specified parameter name and data type.

Syntax

Visual Basic

```
Public Sub HanaParameter (
    ByVal parameterName As String,
    ByVal dbType As HanaDbType
)
```

C#

```
public HanaParameter (
    string parameterName,
    HanaDbType dbType
)
```

Parameters

parameterName

The name of the parameter.

dbType

One of the HanaDbType values.

Related Information

HanaDbType property [page 336]

4.1.4.20.1.3 HanaParameter(string, HanaDbType, int) constructor

Initializes a HanaParameter object with the specified parameter name and data type.

Syntax

Visual Basic

```
Public Sub HanaParameter (
    ByVal parameterName As String,
    ByVal dbType As HanaDbType,
    ByVal size As Integer
)
```

C#

```
public HanaParameter (
    string parameterName,
    HanaDbType dbType,
    int size
)
```

Parameters

parameterName

The name of the parameter.

dbType

One of the HanaDbType values.

size

The length of the parameter.

4.1.4.20.1.4 HanaParameter(string, HanaDbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object) constructor

Initializes a HanaParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source column, source version, and value.

Syntax

Visual Basic

```
Public Sub HanaParameter (
    ByVal parameterName As String,
    ByVal dbType As HanaDbType,
    ByVal size As Integer,
    ByVal direction As ParameterDirection,
    ByVal isNullable As Boolean,
```

```
    ByVal precision As Byte,
    ByVal scale As Byte,
    ByVal sourceColumn As String,
    ByVal sourceVersion As DataRowVersion,
    ByVal value As Object
)
```

C#

```
public HanaParameter (
    string parameterName,
    HanaDbType dbType,
    int size,
    ParameterDirection direction,
    bool isNullable,
    byte precision,
    byte scale,
    string sourceColumn,
    DataRowVersion sourceVersion,
    object value
)
```

Parameters

parameterName

The name of the parameter.

dbType

One of the HanaDbType values.

size

The length of the parameter.

direction

One of the ParameterDirection values.

isNullable

True if the value of the field can be null; false otherwise.

precision

The total number of digits to the left and right of the decimal point to which Value is resolved.

scale

The total number of decimal places to which Value is resolved.

sourceColumn

The name of the source column to map.

sourceVersion

One of the DataRowVersion values.

value

An Object that is the value of the parameter.

4.1.4.20.1.5 HanaParameter(string, HanaDbType, int, string) constructor

Initializes a HanaParameter object with the specified parameter name, data type, and length.

Syntax

Visual Basic

```
Public Sub HanaParameter (
    ByVal parameterName As String,
    ByVal dbType As HanaDbType,
    ByVal size As Integer,
    ByVal sourceColumn As String
)
```

C#

```
public HanaParameter (
    string parameterName,
    HanaDbType dbType,
    int size,
    string sourceColumn
)
```

Parameters

parameterName

The name of the parameter.

dbType

One of the HanaDbType values.

size

The length of the parameter.

sourceColumn

The name of the source column to map.

4.1.4.20.1.6 HanaParameter(string, object) constructor

Initializes a HanaParameter object with the specified parameter name and value.

Syntax

Visual Basic

```
Public Sub HanaParameter (
    ByVal parameterName As String,
    ByVal value As Object
)
```

C#

```
public HanaParameter (
    string parameterName,
    object value
)
```

Parameters

parameterName

The name of the parameter.

value

An Object that is the value of the parameter.

Remarks

This constructor is not recommended; it is provided for compatibility with other data providers.

4.1.4.20.2 ResetDbType() method

Resets the type (the values of DbType and HanaDbType) associated with this HanaParameter.

Syntax

Visual Basic

```
Public Overrides Sub ResetDbType ()
```

C#

```
public override void ResetDbType ()
```

4.1.4.20.3 ToString() method

Returns a string containing the ParameterName.

Syntax

Visual Basic

```
Public Overrides Function ToString () As String
```

C#

```
public override string ToString ()
```

Returns

The name of the parameter.

4.1.4.20.4 DbType property

Gets and sets the DbType of the parameter.

Syntax

Visual Basic

```
Public Overrides Property DbType As DbType
```

C#

```
public override DbType DbType {get;set;}
```

Remarks

The HanaDbType and DbType are linked. Setting the DbType changes the HanaDbType to a supporting HanaDbType.

The value must be a member of the HanaDbType enumerator.

4.1.4.20.5 Direction property

Gets and sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Syntax

Visual Basic

```
Public Overrides Property Direction As ParameterDirection
```

C#

```
public override ParameterDirection Direction {get;set;}
```

Remarks

One of the ParameterDirection values.

If the ParameterDirection is output, and execution of the associated HanaCommand does not return a value, then the HanaParameter object contains a null value. After the last row from the last result set is read, the Output, InputOut, and ReturnValue parameters are updated.

4.1.4.20.6 HanaDbType property

The HanaDbType of the parameter.

Syntax

Visual Basic

```
Public Property HanaDbType As HanaDbType
```

C#

```
public HanaDbType HanaDbType {get;set;}
```

Remarks

The HanaDbType and DbType are linked. Setting the HanaDbType changes the DbType to a supporting DbType.

The value must be a member of the HanaDbType enumerator.

4.1.4.20.7 IsNullable property

Gets and sets a value indicating whether the parameter accepts null values.

Syntax

Visual Basic

```
Public Overrides Property IsNullable As Boolean
```

C#

```
public override bool IsNullable {get;set;}
```

Remarks

This property is true if null values are accepted; otherwise, it is false. The default is false. Null values are handled using the DBNull class.

4.1.4.20.8 Offset property

Gets and sets the offset to the Value property.

Syntax

Visual Basic

```
Public Property Offset As Integer
```

C#

```
public int Offset {get;set;}
```

Remarks

The offset to the value. The default is 0.

4.1.4.20.9 ParameterName property

Gets and sets the name of the HanaParameter.

Syntax

Visual Basic

```
Public Overrides Property ParameterName As String
```

C#

```
public override string ParameterName {get;set;}
```

Remarks

The default is an empty string.

The data provider uses positional parameters that are marked with a question mark (?) instead of named parameters.

4.1.4.20.10 Precision property

Gets and sets the maximum number of digits used to represent the Value property.

Syntax

Visual Basic

```
Public Property Precision As Byte
```

C#

```
public byte Precision {get;set;}
```

Remarks

The value of this property is the maximum number of digits used to represent the Value property. The default value is 0, which indicates that the data provider sets the precision for the Value property.

The Precision property is only used for decimal and numeric input parameters.

4.1.4.20.11 Scale property

Gets and sets the number of decimal places to which Value is resolved.

Syntax

Visual Basic

```
Public Property Scale As Byte
```

C#

```
public byte Scale {get;set;}
```

Remarks

The number of decimal places to which Value is resolved. The default is 0.

The Scale property is only used for decimal and numeric input parameters.

4.1.4.20.12 Size property

Gets and sets the maximum size, in bytes, of the data within the column.

Syntax

Visual Basic

```
Public Overrides Property Size As Integer
```

C#

```
public override int Size {get;set;}
```

Remarks

The value of this property is the maximum size, in bytes, of the data within the column. The default value is inferred from the parameter value.

The Size property is used for binary and string types.

For variable length data types, the Size property describes the maximum amount of data to transmit to the SAP HANA database server. For example, the Size property can be used to limit the amount of data sent to the SAP HANA database server for a string value to the first one hundred bytes.

If Size is not explicitly set, then it is inferred from the actual size of the specified parameter value. For fixed width data types, the value of Size is ignored. It can be retrieved for informational purposes, and returns the maximum amount of bytes the provider uses when transmitting the value of the parameter to the SAP HANA database server.

4.1.4.20.13 SourceColumn property

Gets and sets the name of the source column mapped to the DataSet and used for loading or returning the value.

Syntax

Visual Basic

```
Public Overrides Property SourceColumn As String
```

C#

```
public override string SourceColumn {get;set;}
```

Remarks

A string specifying the name of the source column mapped to the DataSet and used for loading or returning the value.

When SourceColumn is set to anything other than an empty string, the value of the parameter is retrieved from the column with the SourceColumn name. If Direction is set to Input, then the value is taken from the DataSet. If Direction is set to Output, then the value is taken from the data source. A Direction of InputOutput is a combination of both.

4.1.4.20.14 SourceColumnNullMapping property

Gets and sets value that indicates whether the source column is nullable.

Syntax

Visual Basic

```
Public Overrides Property SourceColumnNullMapping As Boolean
```

C#

```
public override bool SourceColumnNullMapping {get;set;}
```

Remarks

This property allows HanaCommandBuilder to generate Update statements for nullable columns correctly.

If the source column is nullable, then true is returned; otherwise, false is returned.

4.1.4.20.15 SourceVersion property

Gets and sets the DataRowVersion to use when loading Value.

Syntax

Visual Basic

```
Public Overrides Property SourceVersion As DataRowVersion
```

C#

```
public override DataRowVersion SourceVersion {get;set;}
```

Remarks

Used by UpdateCommand during an Update operation to determine whether the parameter value is set to Current or Original. This property allows primary keys to be updated. This property is ignored by InsertCommand and DeleteCommand. This property is set to the version of the DataRow used by the Item property, or the GetChildRows method of the DataRow object.

4.1.4.20.16 Value property

Gets and sets the value of the parameter.

Syntax

Visual Basic

```
Public Overrides Property Value As Object
```

C#

```
public override object Value {get;set;}
```

Remarks

An Object that specifies the value of the parameter.

For input parameters, the value is bound to the HanaCommand that is sent to the SAP HANA database server. For output and return value parameters, the value is set on completion of the HanaCommand and after the HanaDataReader is closed.

When sending a null parameter value to the SAP HANA database server, specify DBNull, not null. The null value in the system is an empty object that has no value. DBNull is used to represent null values.

If the application specifies the database type, then the bound value is converted to that type when the data provider sends the data to the SAP HANA database server. The provider attempts to convert any type of value if it supports the IConvertible interface. Conversion errors may result if the specified type is not compatible with the value.

Both the DbType and HanaDbType properties can be inferred by setting the Value.

The Value property is overwritten by Update.

4.1.4.21 HanaParameterCollection class

Represents all parameters to a HanaCommand object and, optionally, their mapping to a DataSet column.

Syntax

Visual Basic

```
Public NotInheritable Class HanaParameterCollection Inherits
System.Data.Common.DbParameterCollection
```

C#

```
public sealed class HanaParameterCollection :
System.Data.Common.DbParameterCollection
```

Members

All members of HanaParameterCollection, including inherited members.

Methods

Modifier and Type	Method	Description
public HanaParameter	Add [page 344]	Adds a HanaParameter object to this collection.
public override void	AddRange [page 350]	Adds an array of values to the end of the HanaParameterCollection.
public HanaParameter	AddWithValue(string, object) [page 351]	Adds a value to the end of this collection.
public override void	Clear() [page 352]	Removes all items from the collection.
public override bool	Contains [page 352]	Indicates whether a HanaParameter object exists in the collection.
public override void	CopyTo(Array, int) [page 354]	Copies HanaParameter objects from the HanaParameterCollection to the specified array.
public override IEnumerator	GetEnumerator() [page 355]	Returns an enumerator that iterates through the HanaParameterCollection.
protected override DbParameter	GetParameter [page 355]	Returns a parameter from the HanaParameterCollection object.
public override int	IndexOf [page 357]	Returns the location of the HanaParameter object in the collection.
public override void	Insert(int, object) [page 359]	Inserts a HanaParameter object in the collection at the specified index.
public override void	Remove(object) [page 360]	Removes the specified HanaParameter object from the collection.

Modifier and Type	Method	Description
public override void	RemoveAt [page 360]	Removes the specified HanaParameter object from the collection.
protected override void	SetParameter [page 362]	Sets a parameter in the HanaParameterCollection object.

Properties

Modifier and Type	Property	Description
public override int	Count [page 364]	Returns the number of HanaParameter objects in the collection.
public override bool	IsFixedSize [page 364]	Gets a value that indicates whether the HanaParameterCollection has a fixed size.
public override bool	IsReadOnly [page 365]	Gets a value that indicates whether the HanaParameterCollection is read-only.
public override bool	IsSynchronized [page 365]	Gets a value that indicates whether the HanaParameterCollection object is synchronized.
public override object	SyncRoot [page 366]	Gets an object that can be used to synchronize access to the HanaParameterCollection.
public new HanaParameter	this [page 366]	Gets and sets the HanaParameter object at the specified index.

Remarks

There is no constructor for HanaParameterCollection. You obtain a HanaParameterCollection object from the HanaCommand.Parameters property of a HanaCommand object.

Related Information

HanaCommand class [page 147]
Parameters property [page 181]
HanaParameter class [page 327]

4.1.4.21.1 Add method

Adds a HanaParameter object to this collection.

Overload list

Modifier and Type	Overload name	Description
public HanaParameter	Add(HanaParameter) [page 344]	Adds a HanaParameter object to this collection.
public override int	Add(object) [page 345]	Adds a HanaParameter object to this collection.
public HanaParameter	Add(string, HanaDbType) [page 346]	Adds a HanaParameter object to this collection, created using the specified parameter name and data type, to the collection.
public HanaParameter	Add(string, HanaDbType, int) [page 347]	Adds a HanaParameter object to this collection, created using the specified parameter name, data type, and length, to the collection.
public HanaParameter	Add(string, HanaDbType, int, string) [page 348]	Adds a HanaParameter object to this collection, created using the specified parameter name, data type, length, and source column name, to the collection.
public HanaParameter	Add(string, object) [page 349]	Adds a HanaParameter object to this collection, created using the specified parameter name and value, to the collection.

4.1.4.21.1.1 Add(HanaParameter) method

Adds a HanaParameter object to this collection.

Syntax

Visual Basic

```
Public Function Add (ByVal value As HanaParameter) As HanaParameter
```

C#

```
public HanaParameter Add (HanaParameter value)
```

Parameters

value

The HanaParameter object to add to the collection.

Returns

The new HanaParameter object.

4.1.4.21.1.2 Add(object) method

Adds a HanaParameter object to this collection.

Syntax

Visual Basic

```
Public Overrides Function Add (ByVal value As Object) As Integer
```

C#

```
public override int Add (object value)
```

Parameters

value

The HanaParameter object to add to the collection.

Returns

The index of the new HanaParameter object.

Related Information

HanaParameter class [page 327]

4.1.4.21.1.3 Add(string, HanaDbType) method

Adds a HanaParameter object to this collection, created using the specified parameter name and data type, to the collection.

Syntax

Visual Basic

```
Public Function Add (
    ByVal parameterName As String,
    ByVal hanaDbType As HanaDbType
) As HanaParameter
```

C#

```
public HanaParameter Add (
    string parameterName,
    HanaDbType hanaDbType
)
```

Parameters

parameterName

The name of the parameter.

hanaDbType

One of the HanaDbType values.

Returns

The new HanaParameter object.

Related Information

HanaDbType enumeration [page 390]
Add(HanaParameter) method [page 344]
Add(string, object) method [page 349]

4.1.4.21.1.4 Add(string, HanaDbType, int) method

Adds a HanaParameter object to this collection, created using the specified parameter name, data type, and length, to the collection.

Syntax

Visual Basic

```
Public Function Add (
    ByVal parameterName As String,
    ByVal hanaDbType As HanaDbType,
    ByVal size As Integer
) As HanaParameter
```

C#

```
public HanaParameter Add (
    string parameterName,
    HanaDbType hanaDbType,
    int size
)
```

Parameters

parameterName

The name of the parameter.

hanaDbType

One of the HanaDbType values.

size

The length of the parameter.

Returns

The new HanaParameter object.

Related Information

HanaDbType enumeration [page 390]
Add(HanaParameter) method [page 344]
Add(string, object) method [page 349]

4.1.4.21.1.5 Add(string, HanaDbType, int, string) method

Adds a HanaParameter object to this collection, created using the specified parameter name, data type, length, and source column name, to the collection.

Syntax

Visual Basic

```
Public Function Add (
    ByVal parameterName As String,
    ByVal hanaDbType As HanaDbType,
    ByVal size As Integer,
    ByVal sourceColumn As String
) As HanaParameter
```

C#

```
public HanaParameter Add (
    string parameterName,
    HanaDbType hanaDbType,
    int size,
    string sourceColumn
)
```

Parameters

parameterName

The name of the parameter.

hanaDbType

One of the HanaDbType values.

size

The length of the column.

sourceColumn

The name of the source column to map.

Returns

The new HanaParameter object.

Related Information

HanaDbType enumeration [page 390]
Add(HanaParameter) method [page 344]
Add(string, object) method [page 349]

4.1.4.21.1.6 Add(string, object) method

Adds a HanaParameter object to this collection, created using the specified parameter name and value, to the collection.

Syntax

Visual Basic

```
Public Function Add (
    ByVal parameterName As String,
    ByVal value As Object
) As HanaParameter
```

C#

```
public HanaParameter Add (
    string parameterName,
    object value
)
```

Parameters

parameterName

The name of the parameter.

value

The value of the parameter to add to the connection.

Returns

The new HanaParameter object.

Remarks

Because of the special treatment of the 0 and 0.0 constants and the way overloaded methods are resolved, explicitly cast constant values to the desired object type when using this method.

Related Information

HanaParameter class [page 327]

4.1.4.21.2 AddRange method

Adds an array of values to the end of the HanaParameterCollection.

Overload list

Modifier and Type	Overload name	Description
public override void	AddRange(Array) [page 350]	Adds an array of values to the end of the HanaParameterCollection.
public void	AddRange(HanaParameter[]) [page 350]	Adds an array of values to the end of the HanaParameterCollection.

4.1.4.21.2.1 AddRange(Array) method

Adds an array of values to the end of the HanaParameterCollection.

Syntax

Visual Basic

```
Public Overrides Sub AddRange (ByVal values As Array)
```

C#

```
public override void AddRange (Array values)
```

Parameters

values

The values to add.

4.1.4.21.2.2 AddRange(HanaParameter[]) method

Adds an array of values to the end of the HanaParameterCollection.

Syntax

Visual Basic

```
Public Sub AddRange (ByVal values As HanaParameter())
```

C#

```
public void AddRange (HanaParameter[] values)
```

Parameters

values

An array of HanaParameter objects to add to the end of this collection.

4.1.4.21.3 AddWithValue(string, object) method

Adds a value to the end of this collection.

Syntax

Visual Basic

```
Public Function AddWithValue (
    ByVal parameterName As String,
    ByVal value As Object
) As HanaParameter
```

C#

```
public HanaParameter AddWithValue (
    string parameterName,
    object value
)
```

Parameters

parameterName

The name of the parameter.

value

The value to be added.

Returns

The new HanaParameter object.

4.1.4.21.4 Clear() method

Removes all items from the collection.

Syntax

Visual Basic

```
Public Overrides Sub Clear ()
```

C#

```
public override void Clear ()
```

4.1.4.21.5 Contains method

Indicates whether a HanaParameter object exists in the collection.

Overload list

Modifier and Type	Overload name	Description
public override bool	Contains(object) [page 352]	Indicates whether a HanaParameter object exists in the collection.
public override bool	Contains(string) [page 353]	Indicates whether a HanaParameter object exists in the collection.

4.1.4.21.5.1 Contains(object) method

Indicates whether a HanaParameter object exists in the collection.

Syntax

Visual Basic

```
Public Overrides Function Contains (ByVal value As Object) As
Boolean
```

C#

```
public override bool Contains (object value)
```

Parameters

value

The HanaParameter object to find.

Returns

True if the collection contains the HanaParameter object; false otherwise.

Related Information

HanaParameter class [page 327]
Contains(string) method [page 353]

4.1.4.21.5.2 Contains(string) method

Indicates whether a HanaParameter object exists in the collection.

Syntax

Visual Basic

```
Public Overrides Function Contains (ByVal value As String) As
Boolean
```

C#

```
public override bool Contains (string value)
```

Parameters

value

The name of the parameter to search for.

Returns

True if the collection contains the HanaParameter object; false otherwise.

Related Information

HanaParameter class [page 327]
Contains(object) method [page 352]

4.1.4.21.6 CopyTo(Array, int) method

Copies HanaParameter objects from the HanaParameterCollection to the specified array.

Syntax

Visual Basic

```
Public Overrides Sub CopyTo (
    ByVal array As Array,
    ByVal index As Integer
)
```

C#

```
public override void CopyTo (
    Array array,
    int index
)
```

Parameters

array

The array to copy the HanaParameter objects into.

index

The starting index of the array.

Related Information

HanaParameter class [page 327]
HanaParameterCollection class [page 342]

4.1.4.21.7 GetEnumerator() method

Returns an enumerator that iterates through the HanaParameterCollection.

Syntax

Visual Basic

```
Public Overrides Function GetEnumerator () As
System.Collections.IEnumerator
```

C#

```
public override IEnumerator GetEnumerator ()
```

Returns

A System.Collections.IEnumerator for the HanaParameterCollection object.

Related Information

HanaParameterCollection class [page 342]

4.1.4.21.8 GetParameter method

Returns a parameter from the HanaParameterCollection object.

Overload list

Modifier and Type	Overload name	Description
protected override DbParameter	GetParameter(int) [page 356]	Returns a parameter from the HanaParameterCollection object.
protected override DbParameter	GetParameter(string) [page 356]	Returns a parameter from the HanaParameterCollection object.

4.1.4.21.8.1 GetParameter(int) method

Returns a parameter from the HanaParameterCollection object.

Syntax

Visual Basic

```
Protected Overrides Function GetParameter (ByVal index As Integer)
As DbParameter
```

C#

```
protected override DbParameter GetParameter (int index)
```

Parameters

index

The zero-based index of the parameter within the collection.

Returns

A System.Data.Common.DbParameter from HanaParameterCollection object.

Related Information

HanaParameterCollection class [page 342]

4.1.4.21.8.2 GetParameter(string) method

Returns a parameter from the HanaParameterCollection object.

Syntax

Visual Basic

```
Protected Overrides Function GetParameter (ByVal parameterName As
String) As DbParameter
```

C#

```
protected override DbParameter GetParameter (string parameterName)
```

Parameters

parameterName

The name of the parameter to locate.

Returns

A System.Data.Common.DbParameter from HanaParameterCollection object.

Related Information

HanaParameterCollection class [page 342]

4.1.4.21.9 IndexOf method

Returns the location of the HanaParameter object in the collection.

Overload list

Modifier and Type	Overload name	Description
public override int	IndexOf(object) [page 357]	Returns the location of the HanaParameter object in the collection.
public override int	IndexOf(string) [page 358]	Returns the location of the HanaParameter object in the collection.

4.1.4.21.9.1 IndexOf(object) method

Returns the location of the HanaParameter object in the collection.

Syntax

Visual Basic

```
Public Overrides Function IndexOf (ByVal value As Object) As
Integer
```

C#

```
public override int IndexOf (object value)
```

Parameters

value

The HanaParameter object to locate.

Returns

The zero-based location of the HanaParameter object in the collection.

Related Information

HanaParameter class [page 327]
IndexOf(string) method [page 358]

4.1.4.21.9.2 IndexOf(string) method

Returns the location of the HanaParameter object in the collection.

Syntax

Visual Basic

```
Public Overrides Function IndexOf (ByVal parameterName As String)
As Integer
```

C#

```
public override int IndexOf (string parameterName)
```

Parameters

parameterName

The name of the parameter to locate.

Returns

The zero-based index of the HanaParameter object in the collection.

Related Information

HanaParameter class [page 327]
IndexOf(object) method [page 357]

4.1.4.21.10 Insert(int, object) method

Inserts a HanaParameter object in the collection at the specified index.

Syntax

Visual Basic

```
Public Overrides Sub Insert (
    ByVal index As Integer,
    ByVal value As Object
)
```

C#

```
public override void Insert (
    int index,
    object value
)
```

Parameters

index

The zero-based index where the parameter is to be inserted within the collection.

value

The HanaParameter object to add to the collection.

4.1.4.21.11 Remove(object) method

Removes the specified HanaParameter object from the collection.

Syntax

Visual Basic

```
Public Overrides Sub Remove (ByVal value As Object)
```

C#

```
public override void Remove (object value)
```

Parameters

value

The HanaParameter object to remove from the collection.

4.1.4.21.12 RemoveAt method

Removes the specified HanaParameter object from the collection.

Overload list

Modifier and Type	Overload name	Description
public override void	RemoveAt(int) [page 360]	Removes the specified HanaParameter object from the collection.
public override void	RemoveAt(string) [page 361]	Removes the specified HanaParameter object from the collection.

4.1.4.21.12.1 RemoveAt(int) method

Removes the specified HanaParameter object from the collection.

Syntax

Visual Basic

```
Public Overrides Sub RemoveAt (ByVal index As Integer)
```

C#

```
public override void RemoveAt (int index)
```

Parameters

index

The zero-based index of the parameter to remove.

Related Information

RemoveAt(string) method [page 361]

4.1.4.21.12.2 RemoveAt(string) method

Removes the specified HanaParameter object from the collection.

Syntax

Visual Basic

```
Public Overrides Sub RemoveAt (ByVal parameterName As String)
```

C#

```
public override void RemoveAt (string parameterName)
```

Parameters

parameterName

The name of the HanaParameter object to remove.

Related Information

RemoveAt(int) method [page 360]

4.1.4.21.13 SetParameter method

Sets a parameter in the HanaParameterCollection object.

Overload list

Modifier and Type	Overload name	Description
protected override void	SetParameter(int, DbParameter) [page 362]	Sets a parameter in the HanaParameterCollection object.
protected override void	SetParameter(string, DbParameter) [page 363]	Sets a parameter in the HanaParameterCollection object.

4.1.4.21.13.1 SetParameter(int, DbParameter) method

Sets a parameter in the HanaParameterCollection object.

Syntax

Visual Basic

```
Protected Overrides Sub SetParameter (
    ByVal index As Integer,
    ByVal value As DbParameter
)
```

C#

```
protected override void SetParameter (
    int index,
    DbParameter value
)
```

Parameters

index

The zero-based index of the parameter to set.

value

A System.Data.Common.DbParameter to be inserted into the HanaParameterCollection object.

Related Information

HanaParameterCollection class [page 342]

4.1.4.21.13.2 SetParameter(string, DbParameter) method

Sets a parameter in the HanaParameterCollection object.

Syntax

Visual Basic

```
Protected Overrides Sub SetParameter (
    ByVal parameterName As String,
    ByVal value As DbParameter
)
```

C#

```
protected override void SetParameter (
    string parameterName,
    DbParameter value
)
```

Parameters

parameterName

The name of the parameter to set.

value

A System.Data.Common.DbParameter to be inserted into the HanaParameterCollection object.

Related Information

GetParameter(string) method [page 356]
IndexOf(object) method [page 357]
IndexOf(string) method [page 358]
RemoveAt(int) method [page 360]
RemoveAt(string) method [page 361]
SetParameter(int, DbParameter) method [page 362]

4.1.4.21.14 Count property

Returns the number of HanaParameter objects in the collection.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property Count As Integer
```

C#

```
public override int Count {get;}
```

Remarks

The number of HanaParameter objects in the collection.

Related Information

HanaParameter class [page 327]
HanaParameterCollection class [page 342]

4.1.4.21.15 IsFixedSize property

Gets a value that indicates whether the HanaParameterCollection has a fixed size.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property IsFixedSize As Boolean
```

C#

```
public override bool IsFixedSize {get;}
```

Remarks

True if this collection has a fixed size; false otherwise.

4.1.4.21.16 IsReadOnly property

Gets a value that indicates whether the HanaParameterCollection is read-only.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property IsReadOnly As Boolean
```

C#

```
public override bool IsReadOnly {get;}
```

Remarks

True if this collection is read-only; false otherwise.

4.1.4.21.17 IsSynchronized property

Gets a value that indicates whether the HanaParameterCollection object is synchronized.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property IsSynchronized As Boolean
```

C#

```
public override bool IsSynchronized {get;}
```

Remarks

True if this collection is synchronized; false otherwise.

4.1.4.21.18 SyncRoot property

Gets an object that can be used to synchronize access to the HanaParameterCollection.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property SyncRoot As Object
```

C#

```
public override object SyncRoot {get;}
```

4.1.4.21.19 this property

Gets and sets the HanaParameter object at the specified index.

Overload list

Modifier and Type	Overload name	Description
public new HanaParameter	this[int index] [page 366]	Gets and sets the HanaParameter object at the specified index.
public new HanaParameter	this[string parameterName] [page 367]	Gets and sets the HanaParameter object at the specified index.

4.1.4.21.19.1 this[int index] property

Gets and sets the HanaParameter object at the specified index.

Syntax

Visual Basic

```
Public Shadows Property Item (ByVal index As Integer) As
HanaParameter
```

C#

```
public new HanaParameter this[int index] {get;set;}
```

Returns

The HanaParameter at the specified index.

Remarks

A HanaParameter object.

In C#, this property is the indexer for the HanaParameterCollection object.

Related Information

HanaParameter class [page 327]
HanaParameterCollection class [page 342]

4.1.4.21.19.2 this[string parameterName] property

Gets and sets the HanaParameter object at the specified index.

Syntax

Visual Basic

```
Public Shadows Property Item (ByVal parameterName As String) As
HanaParameter
```

C#

```
public new HanaParameter this[string parameterName] {get;set;}
```

Returns

The HanaParameter object with the specified name.

Remarks

A HanaParameter object.

In C#, this property is the indexer for the HanaParameterCollection object.

Related Information

HanaParameter class [page 327]
HanaParameterCollection class [page 342]
GetOrdinal(string) method [page 291]
GetValue(int) method [page 296]
GetFieldType(int) method [page 285]
HanaParameter class [page 327]
HanaParameterCollection class [page 342]

4.1.4.22 HanaPermission class

Enables the data provider to ensure that a user has a security level adequate to access an SAP HANA data source. This class is not supported for .NET Core.

Syntax

Microsoft Visual Basic

```
Public NotInheritable Class HanaPermission Inherits
System.Data.Common.DBDataPermission
```

Microsoft C#

```
public sealed class HanaPermission :
System.Data.Common.DBDataPermission
```

Members

All members of HanaPermission, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaPermission(PermissionState) [page 369]	Initializes a new instance of the HanaPermission class.

Methods

Modifier and Type	Method	Description
protected override DBDataPermission	CreateInstance() [page 369]	Creates a new instance of a HanaPermission class.

4.1.4.22.1 HanaPermission(PermissionState) constructor

Initializes a new instance of the HanaPermission class. This constructor is not supported for .NET Core.

Syntax

Visual Basic

```
Public Sub HanaPermission (ByVal state As PermissionState)
```

C#

```
public HanaPermission (PermissionState state)
```

Parameters

state

One of the PermissionState values.

4.1.4.22.2 CreateInstance() method

Creates a new instance of a HanaPermission class. This method is not supported for .NET Core.

Syntax

Visual Basic

```
Protected Overrides Function CreateInstance () As DBDataPermission
```

C#

```
protected override DBDataPermission CreateInstance ()
```

Returns

A new HanaPermission object.

4.1.4.23 HanaPermissionAttribute class

Associates a security action with a custom security attribute.

Syntax

Visual Basic

```
Public NotInheritable Class HanaPermissionAttribute Inherits
System.Data.Common.DBDataPermissionAttribute
```

C#

```
public sealed class HanaPermissionAttribute :
System.Data.Common.DBDataPermissionAttribute
```

Members

All members of HanaPermissionAttribute, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaPermissionAttribute(SecurityAction) [page 370]	Initializes a new instance of the HanaPermissionAttribute class.

Methods

Modifier and Type	Method	Description
public override IPermission	CreatePermission() [page 371]	Returns a HanaPermission object that is configured according to the attribute properties.

4.1.4.23.1 HanaPermissionAttribute(SecurityAction) constructor

Initializes a new instance of the HanaPermissionAttribute class.

Syntax

Visual Basic

```
Public Sub HanaPermissionAttribute (ByVal action As SecurityAction)
```

C#

```
public HanaPermissionAttribute (SecurityAction action)
```

Parameters

action

One of the SecurityAction values representing an action that can be performed using declarative security.

Returns

A HanaPermissionAttribute object.

4.1.4.23.2 CreatePermission() method

Returns a HanaPermission object that is configured according to the attribute properties.

Syntax

Visual Basic

```
Public Overrides Function CreatePermission () As IPermission
```

C#

```
public override IPermission CreatePermission ()
```

4.1.4.24 HanaRowsCopiedEventArgs class

Represents the set of arguments passed to the HanaRowsCopiedEventHandler.

Syntax

Visual Basic

```
Public NotInheritable Class HanaRowsCopiedEventArgs
```

C#

```
public sealed class HanaRowsCopiedEventArgs
```

Members

All members of HanaRowsCopiedEventArgs, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaRowsCopiedEventArgs(long) [page 372]	Creates a new instance of the HanaRowsCopiedEventArgs object.

Properties

Modifier and Type	Property	Description
public bool	Abort [page 372]	Gets or sets a value that indicates whether the bulk-copy operation should be aborted.
public long	RowsCopied [page 373]	Gets the number of rows copied during the current bulk-copy operation.

4.1.4.24.1 HanaRowsCopiedEventArgs(long) constructor

Creates a new instance of the HanaRowsCopiedEventArgs object.

Syntax

Visual Basic

```
Public Sub HanaRowsCopiedEventArgs (ByVal rowsCopied As Long)
```

C#

```
public HanaRowsCopiedEventArgs (long rowsCopied)
```

Parameters

rowsCopied

An 64-bit integer value that indicates the number of rows copied during the current bulk-copy operation.

4.1.4.24.2 Abort property

Gets or sets a value that indicates whether the bulk-copy operation should be aborted.

Syntax

Visual Basic

```
Public Property Abort As Boolean
```

C#

```
public bool Abort {get;set;}
```

4.1.4.24.3 RowsCopied property

Gets the number of rows copied during the current bulk-copy operation.

Syntax

Visual Basic

```
Public ReadOnly Property RowsCopied As Long
```

C#

```
public long RowsCopied {get;}
```

4.1.4.25 HanaRowUpdatedEventArgs class

Provides data for the RowUpdated event.

Syntax

Visual Basic

```
Public NotInheritable Class HanaRowUpdatedEventArgs Inherits
System.Data.Common.RowUpdatedEventArgs
```

C#

```
public sealed class HanaRowUpdatedEventArgs :
System.Data.Common.RowUpdatedEventArgs
```

Members

All members of HanaRowUpdatedEventArgs, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaRowUpdatedEventArgs(DataRow, IDbCommand, StatementType, DataTableMapping) [page 374]	Initializes a new instance of the HanaRowUpdatedEventArgs class.

Properties

Modifier and Type	Property	Description
public new HanaCommand	Command [page 375]	Gets the HanaCommand that is executed when DataAdapter.Update is called.

Modifier and Type	Property	Description
public new int	RecordsAffected [page 375]	Returns the number of rows changed, inserted, or deleted by the execution of the SQL statement.

4.1.4.25.1 HanaRowUpdatedEventArgs(DataRow, IDbCommand, StatementType, DataTableMapping) constructor

Initializes a new instance of the HanaRowUpdatedEventArgs class.

Syntax

Visual Basic

```
Public Sub HanaRowUpdatedEventArgs (
    ByVal row As DataRow,
    ByVal command As IDbCommand,
    ByVal statementType As StatementType,
    ByVal tableMapping As DataTableMapping
)
```

C#

```
public HanaRowUpdatedEventArgs (
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
)
```

Parameters

row

The DataRow sent through an Update.

command

The IDbCommand executed when Update is called.

statementType

One of the StatementType values that specifies the type of query executed.

tableMapping

The DataTableMapping sent through an Update.

4.1.4.25.2 Command property

Gets the HanaCommand that is executed when DataAdapter.Update is called.

Syntax

Visual Basic

```
Public ReadOnly Shadows Property Command As HanaCommand
```

C#

```
public new HanaCommand Command {get;}
```

4.1.4.25.3 RecordsAffected property

Returns the number of rows changed, inserted, or deleted by the execution of the SQL statement.

Syntax

Visual Basic

```
Public ReadOnly Shadows Property RecordsAffected As Integer
```

C#

```
public new int RecordsAffected {get;}
```

Remarks

The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for SELECT statements.

4.1.4.26 HanaRowUpdatingEventArgs class

Provides data for the RowUpdating event.

Syntax

Visual Basic

```
Public NotInheritable Class HanaRowUpdatingEventArgs Inherits
System.Data.Common.RowUpdatingEventArgs
```

C#

```
public sealed class HanaRowUpdatingEventArgs :
System.Data.Common.RowUpdatingEventArgs
```

Members

All members of HanaRowUpdatingEventArgs, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaRowUpdatingEventArgs(DataRow, IDbCommand, StatementType, DataTableMapping) [page 376]	Initializes a new instance of the HanaRowUpdatingEventArgs class.

Properties

Modifier and Type	Property	Description
public new HanaCommand	Command [page 377]	Specifies the HanaCommand to execute when performing the Update.

4.1.4.26.1 HanaRowUpdatingEventArgs(DataRow, IDbCommand, StatementType, DataTableMapping) constructor

Initializes a new instance of the HanaRowUpdatingEventArgs class.

Syntax

Visual Basic

```
Public Sub HanaRowUpdatingEventArgs (
    ByVal row As DataRow,
    ByVal command As IDbCommand,
    ByVal statementType As StatementType,
    ByVal tableMapping As DataTableMapping
)
```

C#

```
public HanaRowUpdatingEventArgs (
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
)
```

Parameters

row

The DataRow to update.

command

The IDbCommand to execute during update.

statementType

One of the StatementType values that specifies the type of query executed.

tableMapping

The DataTableMapping sent through an Update.

4.1.4.26.2 Command property

Specifies the HanaCommand to execute when performing the Update.

Syntax

Visual Basic

```
Public Shadows Property Command As HanaCommand
```

C#

```
public new HanaCommand Command {get;set;}
```

4.1.4.27 HanaSqlTrace class

A set of methods to enable logging of SQL statements, communication packages, and method calls between the data provider client and the SAP HANA database server.

Syntax

Visual Basic

```
Public Class HanaSqlTrace
```

C#

```
public class HanaSqlTrace
```

Members

All members of HanaSqlTrace, including inherited members.

Methods

Modifier and Type	Method	Description
public static void	Disable() [page 378]	Use this method to stop tracing.
public static unsafe void	Enable() [page 379]	Use this method to start tracing.

Modifier and Type	Method	Description
public static Hashtable	GetTraceOptions() [page 379]	Returns a HashTable containing the current set of trace options.
public static void	RemoveTraceOption(string) [page 380]	Removes a trace option from the current set of trace options.
public static void	SetTraceOption(string, string) [page 381]	Sets or replaces a trace option in the current set of trace options.

Remarks

There is no constructor for HanaSqlTrace.

The functionality in this class is based on SQLDBC Trace.

4.1.4.27.1 Disable() method

Use this method to stop tracing.

Syntax

Visual Basic

```
Public Shared Sub Disable ()
```

C#

```
public static void Disable ()
```

Remarks

When this method is called, tracing stops and the trace output file is closed.

Example

Stop all low-level tracing of communications between the client and server:

```
HanaSqlTrace.Disable();
```

The trace output file is closed and tracing stops.

4.1.4.27.2 Enable() method

Use this method to start tracing.

Syntax

Visual Basic

```
Public Shared Sub Enable ()
```

C#

```
public static unsafe void Enable ()
```

Remarks

SetTraceOption must be called before the Enable method is called.

The FILENAME option must be specified before enabling tracing. There is no default filename.

Options only take effect when the Enable method is called. Enable must be called to start tracing.

If one of the SQL, SHORT, LONG, or PACKET options is not set before enabling tracing, no trace file is written.

Example

Enable low-level tracing of communications between the client and server:

```
HanaSqlTrace.Enable();
```

The trace options are enabled, the trace file is created, and logging begins.

4.1.4.27.3 GetTraceOptions() method

Returns a HashTable containing the current set of trace options.

Syntax

Visual Basic

```
Public Shared Function GetTraceOptions () As Hashtable
```

C#

```
public static Hashtable GetTraceOptions ()
```

Remarks

Use this method to get the current set of trace options.

Example

Get the current set of trace options:

```
Hashtable traceOptions = HanaSqlTrace.GetTraceOptions();
```

4.1.4.27.4 RemoveTraceOption(string) method

Removes a trace option from the current set of trace options.

Syntax

Visual Basic

```
Public Shared Sub RemoveTraceOption (ByVal option As String)
```

C#

```
public static void RemoveTraceOption (string option)
```

Parameters

option

A string identifying the option to remove.

Remarks

Use this method to unset (clear) an option.

Example

Remove the PACKET trace option from the current set of trace options:

```
HanaSqlTrace.RemoveTraceOption( "PACKET" );
```

The specified trace option is removed.

4.1.4.27.5 SetTraceOption(string, string) method

Sets or replaces a trace option in the current set of trace options.

Syntax

Visual Basic

```
Public Shared Sub SetTraceOption (
    ByVal option As String,
    ByVal value As String
)
```

C#

```
public static void SetTraceOption (
    string option,
    string value
)
```

Parameters

option

A string identifying the option to set or replace.

value

A string specifying the value of the trace option. Any of "TRUE", "FALSE", "YES", "NO", "1", or "0" can be used for options that accept boolean values. A string representing a number (for example, "4096") must be specified for options that accept numeric values.

Remarks

The following trace options are supported.

Option	Description
FILENAME	This option accepts a string value representing the file path of the trace file including its file name and file type.
LONG	This option accepts a boolean string value that enables or disables long trace. Internal method calls with parameters are logged.
PACKET	This option accepts a boolean string value that enables or disables packet trace. Incoming and outgoing packets are logged.
SQL	This option accepts a boolean string value that enables or disables SQL trace. SQL statements and data are logged.

Trace option names are case-insensitive. Boolean values are case-insensitive.

SetTraceOption must be called before the Enable method is called.

The FILENAME option must be specified before enabling tracing. There is no default filename.

Options only take effect when the Enable method is called. Enable must be called to start tracing.

If none of the SQL, LONG, or PACKET options is set before enabling tracing, no trace file is created.

Unrecognized options are not flagged.

The output format of the trace file is HTML.

Example

Set the FILENAME trace option and add it to the current set of trace options:

```
HanaSqlTrace.SetTraceOption( "FILENAME", "\temp\sqltrace.html" );
```

The FILENAME trace option is set.

Set the SQL trace option and add it to the current set of trace options:

```
HanaSqlTrace.SetTraceOption( "SQL", "TRUE" );
```

The SQL trace option is set.

Set the PACKET trace option and add it to the current set of trace options:

```
HanaSqlTrace.SetTraceOption( "PACKET", "TRUE" );
```

The SQL and PACKET trace options are both set.

Related Information

Enable() method [page 379]

4.1.4.28 HanaTransaction class

Represents a SQL transaction.

Syntax

Visual Basic

```
Public NotInheritable Class HanaTransaction Inherits
System.Data.Common.DbTransaction
```

C#

```
public sealed class HanaTransaction :
System.Data.Common.DbTransaction
```

Members

All members of HanaTransaction, including inherited members.

Methods

Modifier and Type	Method	Description
public override void	Commit() [page 384]	Commits the database transaction.
protected override void	Dispose(bool) [page 384]	
public override void	Rollback() [page 384]	Rolls back a transaction from a pending state.

Properties

Modifier and Type	Property	Description
public new HanaConnection	Connection [page 385]	The HanaConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.
protected override DbConnection	DbConnection [page 385]	Specifies the System.Data.Common.DbConnection object associated with the transaction.
public HanaIsolationLevel	HanaIsolationLevel [page 386]	Specifies the extended isolation level for this transaction.
public override System.Data.IsolationLevel	IsolationLevel [page 386]	Specifies the isolation level for this transaction.

Remarks

There is no constructor for HanaTransaction. To obtain a HanaTransaction object, use one of the BeginTransaction methods. To associate a command with a transaction, use the HanaCommand.Transaction property.

For more information, see Transaction processing.

Related Information

BeginTransaction() method [page 209]

BeginTransaction(HanaIsolationLevel) method [page 210]
Transaction property [page 182]

4.1.4.28.1 Commit() method

Commits the database transaction.

Syntax

Visual Basic

```
Public Overrides Sub Commit ()
```

C#

```
public override void Commit ()
```

4.1.4.28.2 Dispose(bool) method

Syntax

Visual Basic

```
Protected Overrides Sub Dispose (ByVal disposing As Boolean)
```

C#

```
protected override void Dispose (bool disposing)
```

4.1.4.28.3 Rollback() method

Rolls back a transaction from a pending state.

Syntax

Visual Basic

```
Public Overrides Sub Rollback ()
```

C#

```
public override void Rollback ()
```

Remarks

The transaction can only be rolled back from a pending state (after BeginTransaction has been called, but before Commit is called).

4.1.4.28.4 Connection property

The HanaConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Syntax

Visual Basic

```
Public ReadOnly Shadows Property Connection As HanaConnection
```

C#

```
public new HanaConnection Connection {get;}
```

Remarks

A single application can have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by BeginTransaction.

4.1.4.28.5 DbConnection property

Specifies the System.Data.Common.DbConnection object associated with the transaction.

Syntax

Visual Basic

```
Protected ReadOnly Overrides Property DbConnection As DbConnection
```

C#

```
protected override DbConnection DbConnection {get;}
```

Returns

The System.Data.Common.DbConnection object associated with the transaction.

4.1.4.28.6 HanaIsolationLevel property

Specifies the extended isolation level for this transaction.

Syntax

Visual Basic

```
Public ReadOnly Property HanaIsolationLevel As HanaIsolationLevel
```

C#

```
public HanaIsolationLevel HanaIsolationLevel {get;}
```

Remarks

The HanaIsolationLevel for this transaction. This can be one of:

- ReadUncommitted
- ReadCommitted
- RepeatableRead
- Serializable

The default is ReadCommitted.

Parallel transactions are not supported. Therefore, the HanaIsolationLevel applies to the entire transaction.

If ReadUncommitted is specified, then ReadCommitted is used instead.

4.1.4.28.7 IsolationLevel property

Specifies the isolation level for this transaction.

Syntax

Visual Basic

```
Public ReadOnly Overrides Property IsolationLevel As
System.Data.IsolationLevel
```

C#

```
public override System.Data.IsolationLevel IsolationLevel {get;}
```

Remarks

The IsolationLevel for this transaction. This can be one of:

- Unspecified
- Chaos
- ReadUncommitted
- ReadCommitted
- RepeatableRead
- Serializable
- Snapshot

The default is ReadCommitted. Unspecified, Chaos, and Snapshot are not supported.

4.1.4.29 HanaInfoMessageEventHandler(object, HanaInfoMessageEventArgs) delegate

Represents the method that handles the HanaConnection.InfoMessage event of a HanaConnection object.

Syntax

Visual Basic

```
Public Delegate Sub HanaInfoMessageEventHandler (
    ByVal obj As Object,
    ByVal args As HanaInfoMessageEventArgs
) As delegate void
```

C#

```
public delegate void HanaInfoMessageEventHandler (
    object obj,
    HanaInfoMessageEventArgs args
);
```

Related Information

HanaConnection class [page 203]
InfoMessage event [page 229]

4.1.4.30 HanaRowsCopiedEventHandler(object, HanaRowsCopiedEventArgs) delegate

Represents the method that handles the HanaBulkCopy.HanaRowsCopied event of a HanaBulkCopy.

Syntax

Visual Basic

```
Public Delegate Sub HanaRowsCopiedEventHandler (
    ByVal sender As Object,
    ByVal rowsCopiedEventArgs As HanaRowsCopiedEventArgs
) As delegate void
```

C#

```
public delegate void HanaRowsCopiedEventHandler (
    object sender,
    HanaRowsCopiedEventArgs rowsCopiedEventArgs
);
```

Related Information

HanaBulkCopy class [page 110]

4.1.4.31 HanaRowUpdatedEventHandler(object, HanaRowUpdatedEventArgs) delegate

Represents the method that handles the RowUpdated event of a HanaDataAdapter.

Syntax

Visual Basic

```
Public Delegate Sub HanaRowUpdatedEventHandler (
    ByVal sender As Object,
    ByVal e As HanaRowUpdatedEventArgs
) As delegate void
```

C#

```
public delegate void HanaRowUpdatedEventHandler (
    object sender,
    HanaRowUpdatedEventArgs e
);
```

4.1.4.32 HanaRowUpdatingEventHandler(object, HanaRowUpdatingEventArgs) delegate

Represents the method that handles the RowUpdating event of a HanaDataAdapter.

Syntax

Visual Basic

```
Public Delegate Sub HanaRowUpdatingEventHandler (
    ByVal sender As Object,
    ByVal e As HanaRowUpdatingEventArgs
) As delegate void
```

C#

```
public delegate void HanaRowUpdatingEventHandler (
    object sender,
    HanaRowUpdatingEventArgs e
);
```

4.1.4.33 HanaBulkCopyOptions enumeration

A bitwise flag that specifies one or more options to use with an instance of HanaBulkCopy.

Syntax

Visual Basic

```
Public Enum HanaBulkCopyOptions
```

C#

```
enum HanaBulkCopyOptions
```

Members

Member name	Description	Value
Default	Specifying only this value causes the default behavior to be used. By default, triggers are enabled.	0x0
TableLock	When specified the table is locked using the command LOCK TABLE table_name WITH HOLD IN SHARE MODE. This lock is in place until the connection is closed.	0x1

Member name	Description	Value
UseInternalTransaction	When specified, each batch of the bulk-copy operation is executed within a transaction. When not specified, transaction aren't used. If you indicate this option and also provide a HanaTransaction object to the constructor, then a System.ArgumentException occurs.	0x2

Remarks

The HanaBulkCopyOptions enumeration is used when you construct a HanaBulkCopy object to specify how the WriteToServer methods behave.

The CheckConstraints and KeepNulls options are not supported.

Related Information

HanaBulkCopy class [page 110]

4.1.4.34 HanaDbType enumeration

Enumerates the SAP HANA database data types for .NET.

Syntax

Visual Basic

```
Public Enum HanaDbType
```

C#

```
enum HanaDbType
```

Members

Member name	Description
BigInt	Signed 64-bit integer.

Member name	Description
Date	The DATE data type consists of year, month, and day information to represent a date value.
Decimal	Exact numerical data, with a specified precision and scale.
Double	Double-precision floating-point number (8 bytes).
Integer	Signed 32-bit integer.
NClob	Large unicode character data.
NVarChar	Variable-length unicode character set string with maximum length of 5000.
Real	Single-precision 32-bit floating-point number.
SecondDate	The SECONDDATE data type consists of year, month, day, hour, minute and second information to represent a date with time value.
ShortText	Variable-length character string that supports text search features and string search features.
SmallDecimal	The SMALLDECIMAL floating-point data type.
SmallInt	Signed 16-bit integer (-32768 - 32767).
Time	The TIME data type consists of hour, minute, and second to represent a time value.
TimeStamp	The TIMESTAMP data type consists of date and time information.
TinyInt	Unsigned 8-bit integer (0 - 255).
VarBinary	Binary data with maximum length of 5000.
TableType	User-defined table type.

Remarks

The table below lists which .NET types are compatible with each HanaDbType. In the case of integral types, table columns can always be set using smaller integer types, but can also be set using larger types as long as the actual value is within the range of the type.

HanaDbType	Compatible .NET type	C# built-in type	Visual Basic built-in type
BigInt	System.Int64	long	Long
Blob	System.Array	Byte[]	Byte()
Date	System.DateTime	DateTime (no built-in type)	Date
Decimal	System.Decimal	decimal	Decimal
Double	System.Double	double	Double
Integer	System.Int32	int	Integer

HanaDbType	Compatible .NET type	C# built-in type	Visual Basic built-in type
NClob	System.String	string	String
NVarChar	System.String	string	String
Real	System.Single	float	Single
SecondDate	System.DateTime	DateTime (no built-in type)	Date
SmallDecimal	System.Decimal	decimal	Decimal
SmallInt	System.Int16	short	Short
Time	System.TimeSpan	TimeSpan (no built-in type)	TimeSpan (no built-in type)
TimeStamp	System.DateTime	DateTime (no built-in type)	Date
TinyInt	System.Byte	byte	Byte
VarBinary	System.Array	Byte[]	Byte()

Binary columns of length 16 are fully compatible with the UniqueIdentifier type.

Related Information

GetFieldType(int) method [page 285]
GetDataTypeName(int) method [page 280]

4.1.4.35 HanaIsolationLevel enumeration

Specifies SAP HANA database isolation levels.

Syntax

Visual Basic

```
Public Enum HanaIsolationLevel
```

C#

```
enum HanaIsolationLevel
```

Members

Member name	Description	Value
ReadUncommitted	This isolation level is not supported by SAP HANA. READ COMMITTED is used instead.	0
ReadCommitted	Sets the isolation level to READ COMMITTED.	1
RepeatableRead	Sets the isolation level to REPEATABLE READ.	2
Serializable	Sets the isolation level to SERIALIZABLE.	3

Remarks

This class augments the System.Data.IsolationLevel class.

The data provider supports all SAP HANA isolation levels. BeginTransaction has been overloaded so it can take either an IsolationLevel or a HanaIsolationLevel. There is a new property in HanaTransaction called HanaIsolationLevel that gets the HanaIsolationLevel.

See the SAP HANA SQL Reference for more information.

4.1.4.36 HanaMessageType enumeration

Identifies the type of message.

Syntax

Visual Basic

```
Public Enum HanaMessageType
```

C#

```
enum HanaMessageType
```

Members

Member name	Description	Value
Action	Message of type ACTION.	2

Member name	Description	Value
Info	Message of type INFO.	0
Status	Message of type STATUS.	3
Warning	Message of type WARNING.	1

Remarks

This can be one of: Action, Info, Status, or Warning.

4.1.4.37 HanaDecimal structure

Represents an SAP HANA decimal value.

Syntax

Visual Basic

```
Structure HanaDecimal
```

C#

```
typedef struct HanaDecimal
```

Members

All members of HanaDecimal, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	HanaDecimal [page 395]	Initializes a HanaDecimal object with a .NET decimal value.

Methods

Modifier and Type	Method	Description
public override bool	Equals(object) [page 396]	
public override int	GetHashCode() [page 397]	Serves as a hash function for the HanaDecimal object.
public Decimal	ToDecimal() [page 397]	Returns a .NET decimal value for this HanaDecimal object.

Modifier and Type	Method	Description
public override String	ToString() [page 398]	Returns the string representation for this HanaDecimal object.

4.1.4.37.1 HanaDecimal constructor

Initializes a HanaDecimal object with a .NET decimal value.

Overload list

Modifier and Type	Overload name	Description
public	HanaDecimal(Decimal) [page 395]	Initializes a HanaDecimal object with a .NET decimal value.
public	HanaDecimal(string) [page 396]	Initializes a HanaDecimal object with a string value.

4.1.4.37.1.1 HanaDecimal(Decimal) constructor

Initializes a HanaDecimal object with a .NET decimal value.

Syntax

Visual Basic

```
Public Sub HanaDecimal (ByVal decimalValue As Decimal)
```

C#

```
public HanaDecimal (Decimal decimalValue)
```

Parameters

decimalValue

A .NET decimal value.

4.1.4.37.1.2 HanaDecimal(string) constructor

Initializes a HanaDecimal object with a string value.

Syntax

Visual Basic

```
Public Sub HanaDecimal (ByVal stringRepresentation As String)
```

C#

```
public HanaDecimal (string stringRepresentation)
```

Parameters

stringRepresentation

The string representation of the HanaDecimal object.

4.1.4.37.2 Equals(object) method

Syntax

Visual Basic

```
Public Overrides Function Equals (ByVal obj As Object) As Boolean
```

C#

```
public override bool Equals (object obj)
```

Parameters

obj

The object to compare with the current object.

Returns

True if the specified object is equal to the current HanaDecimal object; false otherwise.

Remarks

Determines whether the specified System.Object is equal to the current HanaDecimal object.

4.1.4.37.3 GetHashCode() method

Serves as a hash function for the HanaDecimal object.

Syntax

Visual Basic

```
Public Overrides Function GetHashCode () As Integer
```

C#

```
public override int GetHashCode ()
```

Returns

A hash code for the current HanaDecimal object.

4.1.4.37.4 ToDecimal() method

Returns a .NET decimal value for this HanaDecimal object.

Syntax

Visual Basic

```
Public Function ToDecimal () As Decimal
```

C#

```
public Decimal ToDecimal ()
```

Returns

A .NET decimal value.

Remarks

This method throws a ConversionException if the HanaDecimal object cannot be converted to a .NET decimal value.

4.1.4.37.5 ToString() method

Returns the string representation for this HanaDecimal object.

Syntax

Visual Basic

```
Public Overrides Function ToString () As String
```

C#

```
public override String ToString ()
```

Returns

A String value.

4.2 Go (golang) Application Programming

Go (also known as golang) is an open source programming language. The SAP HANA client includes a dedicated Go driver interface that allows Go applications to access the SAP HANA server by using the Go SQL API.

The Go driver is not a pure Go driver: It uses cgo. cgo enables you to create golang packages that call C code and it allows Go programs to interoperate with C libraries. The SAP HANA Go driver is based on SQLDBC and implements the `database/sql/driver` interface. The driver loads the libdbcapiHDB shared DLL. Write your application with the golang SQL API.

For a list of supported Go versions, see 3165810

SAP HANA Go Driver Limitations

The SAP HANA Go driver does not support the following features that are available in the Go API:

Feature	SAP HANA Go Driver Description	
Named parameters	There is no support for named parameters. You can pass a NamedArg as an argument to Query or Exec to bind to the corresponding named parameters. Pass the NamedArg values in the order that the parameters are specified in the SQL.	
Statement timeouts	You can only set the timeout once a statement has been prepared.	
	PrepareContext(ctx.Context, string) function	Use Prepare(string) instead. Once the statement is prepared, set the context timeout. `statement_examples.go` shows how to set a timeout on a prepared statement.
	ExecContext(ctx.Context, string, interface{}) function on DB and Tx	Use Exec(string, interface{}) or ExecContext instead on Stmt instead, which respects the context timeout.
	QueryContext(ctx.Context, string, interface{}) function on DB and Tx	Use Query(string, interface{}) or QueryContext on Stmt instead, which respects the context timeout.
	QueryRowContext(ctx.Context, string, interface{}) function on DB and Tx	Use QueryRow(string, interface{}) or QueryRowContext on Stmt instead, which respects the context timeout.
Context timeouts for pings	The context timeout that is passed to PingContext(ctx.Context) is ignored.	
Read-only transactions	You cannot start a read-only transaction. Setting the ReadOnly option to true in TxOptions results in a "Setting of read only options is currently unsupported" error.	
LastInsertId()	There is no support for LastInsertId(). Calling this method results in a "Feature not supported error".	

Related Information

SAP Note 3136015
The Go Programming Language: Command cgo
The Go Programming Language: Package sql
The Go Programming Language: Package driver
Connect Using the SAP HANA Go Interface on Windows Tutorial

4.2.1 Go (golang) Differences from Other Database Vendors

Differences from other database vendors must be accounted for in your application design.

Timestamp Support

When a time.Time object is provided as an input parameter to a prepared upsert statement, then by default the SAP HANA golang driver (`SAP/go-hdb/driver`) converts the time.Time value to UTC before insertion. This behavior is different from other vendor's corresponding TIMESTAMP WITHOUT TIMEZONE column types: most other vendors drop the timezone information before insertion without adjusting the date and time information.

Use the SAP HANA golang driver's `utcTimeConversion` connection property to change the default behavior. The default behavior is TRUE.

```
db, err := sql.Open("hdb", "hdb://<username>:<password>@<hostname>:<port>?
utcTimeConversion=false")
```

Related Information

Go Connection Properties [page 407]

4.2.2 Set Up Your Application to Use the Go Driver Package

Use the golang driver to connect to an SAP HANA database.

Prerequisites

> **Note**
>
> There are two ways that the Golang driver can be used from a Go application, as a module (recommended) or as a regular package with modules disabled (not recommended).

- The Go driver uses cgo. On 64-bit Microsoft Windows, you must have a gcc compiler installed and the path to it set in your PATH environment variable. The MinGW gcc compiler is recommended for installation.
- You have installed the SAP HANA client. The Go driver source files and examples are located in `<client-install-dir>/golang`.
- The Go driver supports versions 1.8–1.20.

Context

The SAP HANA client install includes source code that you use to build the Go driver.

You can build the driver on its own, or build the driver as part of building the Go examples.

Procedure

Using the Golang Driver Package as a Module (Recommended)

1. Navigate to the golang driver package's installation driver folder and run the following commands to make it recognizable as a Go module:
   - On **Microsoft Windows**:

     ```
     cd <gclient-install-dir>\golang\SAP\go-hdb\driver
     ```

   - On **Linux**:

     ```
     cd <client-install-dir>/golang/SAP/go-hdb/driver
     go mod init “SAP/go-hdb/driver
     ```

2. Specify the golang driver's package location in the application.
   - You must have already made your application a module. If you have not, execute `go mod init <unique-path-name>` in the root folder.
     - If `go.mod` does not already exist, execute `go mod tidy`

- Open `go.mod` in an editor and:
  - Replace `SAP/go-hdb/driver v0.1.0` with `<golang-client-install-dir>\golang\SAP\go-hdb\driver`. Add this line right after the Go `<version>` line (leave a space between the go `<version>` line and the new line).
  - Complete one of the following steps:
    - Add the following line to the end of the require block, if it already exists:

      ```
      SAP/go-hdb/driver v0.1.0
      ```

    - If the require block does not aready exist, add the following line:

      ```
      require SAP/go-hdb/driver v0.1.0
      ```

- Set the CGO_LDFLAGS environment variable to point to the location of the SAP HANA database client library:
  - On **Microsoft Windows**: `set CGO_LDFLAGS=<path-to-dbcapi-library>\libdbcapiHDB.dll`
  - On **MacOS**: `export CGO_LDFLAGS=<path-to-dbcapi-library>/libdbcapiHDB.dylib`
  - On **Linux**: `export CGO_LDFLAGS=<path-to-dbcapi-library>/libdbcapiHDB.so`
- Complete one of the following steps:
  - Place the dbcapi shared library file in the same directory as the application executable
  - Add the library to the loader's path so that cgo can find dbcapi when building the driver:
    - On **Microsoft Windows**: `set PATH=<path-to-dbcapi-library>`
    - On **Linux**: `export LD_LIBRARY_PATH=<path-to-dbcapi-library>`

Using the Golang Driver Package as A Package with Modules Disabled (not recommended)

3. Disable modules for the golang runtime (applicable for Go version 1.11 and later) using the following environment variables:
   - On **Microsoft Windows**: `set GO111MODULE=off`
   - On **Linux**: `export GO111MODULE=off`
4. Specify the golang driver's package location in the application.
5. Add the golang driver's location to the GOPATH (the GOPATH environment variable lists places to look for Go code/packages).

> **Note**
>
> You must keep this set permanently in your application development environment.

   - **Microsoft Windows** (command prompt):

     ```
     go env GOPATH(Note the location [e.g. <the-path>)
     set GOPATH="<the-path>;<client-install-dir>\golang\"
     ```

   - **Linux**:

     ```
     export GOPATH="$(go env GOPATH):<client-install-dir>/golang"
     ```

6. Copy the golang installation into the default GOPATH.
   - **Microsoft Windows** (command prompt):

     ```
     go env GOPATH(Note the location [e.g. <the-path>)
     ```

```
xcopy /E /I <client-install-dir>\golang\SAP <the-path>\src
```

- **Linux**:

```
ln -s <client-install-dir>/golang/SAP $(go env GOPATH)/SAP
```

7. Set the CGO_LDFLAGS environment variable to point to the location of the SAP HANA client library:
   - On **Microsoft Windows**: `set CGO_LDFLAGS=<path-to-dbcapi-library>\libdbcapiHDB.dll`
   - On **MacOS**: `CGO_LDFLAGS=<path-to-dbcapi-library>/libdbcapiHDB.dylib`
   - On **Linux**: `export CGO_LDFLAGS=<path-to-dbcapi-library>/libdbcapiHDB.so`
8. Complete one of the following steps:
   - Place the dbcapi shared library files in the same directory as the application executable.
   - Add the library to the loader's path so that the cgo can find dbcapi, when building the driver:
     - On **Microsoft Windows**: `set PATH=<path-to-dbcapi-library>`
     - On **Linux**: `export LD_LIBRARY_PATH=<path-to-dbcapi-library>`

Related Information

SAP Note 3136015
The Go Programming Language: Getting Started
The Go Programming Language: GOPATH environment variable
Build and Run the Go Examples [page 403]
Connect Using the SAP HANA Go Interface on Windows Tutorial

4.2.3 Build and Run the Go Examples

Use the examples that are included in the client install to learn more about the Go driver.

Prerequisites

- You have set up your Go workspace, including setting the GOPATH and GOROOT environment variables.
- You have added the libdbcapiHDB library to your path or put it in the same folder as the executable for the example.

Context

The client installer includes three examples that show how to use the Go driver:

- `statement_examples.go` prepares a statement, executes it, and then processes the result set.

- `lob_examples.go` provides an example of working with LOBs.
- `transaction_examples.go` starts and ends transactions, as well as prepares, executes, and processes a result set within a transaction.
  The Go driver is automatically built as part of building the examples. If an up-to-date driver is available, then the driver is not built.

Procedure

1. Build the example:

Example	Command
`statement_examples.go`	`go build SAP/go-hdb/driver/examples/statement` An executable that is named `statement` (UNIX and macOS) or `statements.exe` (Microsoft Windows) is added to the folder where you ran the Go build.
`lob_examples.go`	`go build SAP/go-hdb/driver/examples/lobs` An executable that is named `lobs` (UNIX and macOS) or `lobs.exe` (Microsoft Windows) is added to the folder where you ran the Go build.
`transaction_examples.go`	`go build SAP/go-hdb/driver/examples/transactions` An executable that is named `transactions` (UNIX and macOS) or `transactions.exe` (Microsoft Windows) is added to the folder where you ran the Go build.

2. Ensure that libdbcapiHDB library is in the same folder as the executable if you have not added the library to your path.
3. Run the executable to use the examples:

Example	Command
`statement_examples.go`	`statement -dsn hdb://<user>:<password>@<host>:<port>`
`lob_examples.go`	`lobs -dsn hdb://<user>:<password>@<host>:<port>`
`transaction_examples.go`	`transactions -dsn hdb://<user>:<password>@<host>:<port>`

Replace <user> and <password> with the database user and password, and replace <host>:<port> with the SAP HANA server host and port.

4.2.4 Connect from Go to SAP HANA

Use the Go driver to connect to an SAP HANA database.

Prerequisites

You have built the Go driver.

Context

The SAP HANA driver for Go supports all of the SQLDBC connection properties.

You can trace Go connections by using SQLDBC tracing.

Procedure

Connect to the SAP HANA database:

Connection Type	Command
Connection string	`hdb://<user>:<password>@<host>:<port>?<connection-property>=<value>&<connection-property>=<value>&...` Replace <user> and <password> with the database user and password, and replace <host>:<port> with the SAP HANA server host and port. hdb://<user>:<password>@<host>:<port> is a valid connection string The <connection-property> names are case sensitive. Connection properties are not required. If a property value contains URL special characters (such as providing the full certificate string as the value for the

Connection Type	Command

	sslTrustStore connect property), URL encode its value. Use the net/url package to do this in one of two ways:
	1. Quoting the specific value:
	`encodedValue := url.QueryEscape("<original value>")`
	2. Building the entire string via the package:
	`connectURL, err := url.Parse(""hdb://<username>:<password>@<hostname>:<port>)`
	`if err != nil {`
	`// Report error and return`
	`}`
	`params := url.Values{}`
	`// For each pair of connectProperty=value add like this`
	`params.Add("<connectProperty>", "<raw value>")`
	`//...`
	`// Add the parameters to the URL`
	`connectURL.RawQuery = params.Encode()`
	`// The final connect string to be used:`
	`connectString = connectURL.String()`

Tenant database in an MDC system	`hdb://<user>:<password>@<host>:<port>?databaseName=<tenant-db-name>` Replace `<user>` and `<password>` with the database user and password, and replace `<host>:<port>` with the SAP HANA server host and port. Replace `<tenant-db-name>` with the name of the tenant database.
hdbuserstore key	`hdb://?key=<keyname>`

Related Information

SQLDBC Tracing [page 785]
Set Up Your Application to Use the Go Driver Package [page 401]

4.2.4.1 Go Connection Properties

When connecting to an SAP HANA database using Go, you can specify Go connection properties or SQLDBC connection properties.

> **Note**
>
> Do not set connection properties to an empty string. The client will substitute any empty string values with the property's default setting.

The following table lists the SAP HANA client-specific Go connection properties (property names are case insensitive):

Property	Value	Default	Description
`charset`	`<character-set>`		Specifies the character set. This value must be either "ASCII", "UTF-8", or "UCS2".
`database`	`<name>`		Specifies the name of the database to connect to.
`lobChunkSize`	`<size>`		Specifies the chunk size of the data that is retrieved and sent by the driver. The default size is the same as the default packet size.
`PWD`	`<password>`		Specifies the password for the SAP HANA database user.
`severNode`	`<host>:<port>[, <host>:<port>]`		Specifies the SAP HANA server address or addresses. If your system contains a failover standby node, you can specify it as an additional optional address.
`UID`	`<user-ID>`		Specifies the name of the SAP HANA database user.
`utcTimeConversion`	Boolean	TRUE	Controls whether or not time.Time objects used as input parameters should be converted to UTC before upserts are executed: TRUE/FALSE.

SQLDBC Connection Properties

The following table lists SQLDBC connection properties (property names are case insensitive) that can be used with Go:

> **Note**
>
> A valid setting for a Boolean property can be: “1”, “TRUE”, “YES”, or “ON” for a TRUE setting or “0”, “FALSE”, “NO”, or “OFF” for a FALSE setting. The setting of a Boolean property is case-insensitive.

Property	Value	Default	Description
`abapVarCharMode`	Boolean	FALSE	Specifies that in ABAP VARCHAR mode, a single space and an empty string are considered the same.
`allowLocalCompress`	Boolean	TRUE	Determines whether or not connections from the same machine as the SAP HANA server are compressed. The `compress` parameter and/or the `indexserver.ini` > *session* > *compression* server configuration control whether or not network compression is enabled. **Note** If the connection uses a proxy, do not use `allowLocalCompress`=FALSE as the connection may not be compressed when required.
`allowReconnectOnSelect`	Boolean	OFF	Allows reconnects to happen for SELECT queries which fail during a receive. By default, this property is set to OFF, which allows for backward compatibility.

Property	Value	Default	Description
`bindAddress`	`<ip-address>`	NULL	Specifies that the provided address is used as the source address for the socket connection. By default, the address preferred by the OS is used. If setting `bindAddress` to a hostname, that hostname should only resolve to a single IP address. In the majority of network configurations, specifying `bindAddress` is not required. **Note** Only specify `bindAddress` if it is required to connect in your network configuration.
`chopBlanks`	Boolean	FALSE	Removes trailing blanks from character output.
`chopBlanksInput`	Boolean	FALSE	Removes trailing blanks from character input.
`communicationTimeout`	Timeout in milliseconds	0 (disabled)	Aborts communication attempts to the server after the specified timeout. If set to a non-zero value, the driver will wait for a maximum of the specified number of milliseconds for any request to the server to complete. If the time has expired and a response has not been received, the driver returns an error.

Property	Value	Default	Description
`compress`	Boolean	FALSE	Enables or disables network compression for the connection. The `indexserver.ini` > *session* > *compression* server configuration defaults to FALSE. Setting this parameter overrides the setting of the `indexserver.ini` > *session* > *compression* parameter. The `allowLocalCompress` property controls if clients running on the same machine as the server use network compression when compression is requested.
`connDownRollbackError`	Boolean	FALSE	Setting this property to TRUE specifies that if the database connection is down due to network or SAP HANA server problems, the client rolls back the last transaction and sends and error to the application. The application can resend the last transaction to the server after properly handling the error. The following conditions must be true: • The session is recoverable in the server • There is a written transaction (uncommitted operation) in the session • The client failed to send the last SQL statement, or the last statement was sent to the server successfully, but both auto-commit and DDL auto-commit in the client are off. By default, this property is set to FALSE.

Property	Value	Default	Description
`connectTimeout`	Timeout in milliseconds	0 (limited by the `nodeConnectTimeout` connection property)	Aborts connection attempts after the specified timeout. `connectTimeout` limits the overall time for the entire connection to try all server addresses specified. See `nodeConnectTimeout` for per-node timeout.
`currentSchema`	`<schema>`	CURRENTUSER	Sets the current schema, which is used for identifiers without a schema.
`cursorHoldabilityType`	NONE, COMMIT, ROLLBACK, COMMIT_AND_ROLLBACK	COMMIT	Specifies whether the cursor for a result set remains open after the transaction is committed or rolled back. If a cursor remains open, then the application is able to fetch more rows from it. NONE means the cursor is always closed after a commit or rollback, while COMMIT means the cursor remains valid after a commit, but not after rollback. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources. Support for holding a cursor over ROLLBACK or COMMIT_AND_ROLLBACK is available in SAP HANA servers beginning with SAP HANA 2.4.

Property	Value	Default	Description
`emptyTimestampIsNull`	Boolean	TRUE	When enabled, DATE, TIME, SECONDDATE, and TIMESTAMP values inserted as empty strings are returned as NULLs. When disabled, these values are returned as the following out-of-band values: • `DATE` - one day before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 00:00.000000000 UTC BC • `TIME` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `SECONDDATE` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `TIMESTAMP` - one hundred nanoseconds before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.999999900 UTC BC

Property	Value	Default	Description
`ignoreTopology`	Boolean	FALSE	Controls how the client uses the topology returned from the server. TRUE indicates to use the topology unless port-forwarding is detected and FALSE indicates to always ignore the topology. It is an error to specify both ignoreTopology=TRUE and siteType=PRIMARY or SECONDARY.
`isolationLevel`	TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE	TRANSACTION_READ_COMMITTED	Sets the isolation level for the connection.
`isoTimestampOutput`	`<string>`	FALSE	Displays whether or not 'T' is used as the separator between the Date and Time parts of a timestamp retrieved as a string: TRUE/FALSE. If FALSE, then a space character is used as the separator.
`key`	`<username>`	Empty	Specifies the key for the hdbuserstore.
`maxLazyDroppedStatements`	`<number>`	50	Controls the maximum number of queued statements to be dropped. Setting this value to 0 disables the lazy drop feature and causes a request to be sent to the server each time a statement is dropped by the application.
`maxVersion`	`<string>`		Specifies the maximum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.

Property	Value	Default	Description
`minVersion`	`<string>`		Specifies the minimum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.
`nodeConnectTimeout`	Timeout in milliseconds	60000	Aborts the connection attempt after the specified maximum timeout. The connection attempt typically fails immediately if the server cannot be found. If connection attempts to a particular server address are slow to fail, then using this connection property reduces the time before a successful connection by minimizing the time waiting to connect to individual addresses. On a host with multiple resolved IP addresses, the network timeout for each attempt to connect to an IP address will be equal to the `nodeConnectTimeout` divided by the number of resolved IP addresses. If this option is set to zero, the `connectTimeout` setting will be used. If both of the `nodeConnectTimeout` and `connectTimeout` are set to zero, then SQLDBC will use the system TCP/IP socket connect timeout to connect to the IP address.

Property	Value	Default	Description
`packetCaching`	Boolean	TRUE	Caching packets reduces the need to continually allocate and deallocate memory for packets. However, this increases the peak memory usage of the application, and for how long it stays at or near that peak.
`packetSize`	`<number>` in bytes	1048576 (1 MB)	Sets the maximum size of a request packet sent from the client to the server, in bytes. In some cases, request packets can be larger (for example, if a SQL statement or parameter value is larger than the packetSize). The minimum is 1 MB.
`packetSizeLimit`	`<number>` in bytes	2147483647 (2 GB)	Sets the maximum size for a request or reply packet, in bytes. If SQLDBC needs to allocate a packet larger than the limit, then it returns an error. If a reply packet is too large, then the connection is dropped. The minimum value is the packetSize value; the maximum value is 2 GB.

Property	Value	Default	Description
`prefetch`	Boolean	TRUE	Controls result set prefetching. When enabled, result set prefetch requests the next block of rows of a result set while the application is still processing the current block of rows. When fetching large result sets, result set prefetch can improve performance; however, it can also result in the client using more memory. In some cases, the server JobExecutor thread could be blocked from sending the reply for an indefinite period of time until the client receives the prefetch reply. If the result set contains LOB columns, then result set prefetch is always disabled.
`{ proxyHostname | proxy_host }`	`<host-name>`		Specifies the host name of the proxy server. The default proxy server is SOCKS5. If `proxyHttp` is set to true, the proxy server is assumed to be an HTTP proxy.
`proxyHttp`	Boolean	FALSE	When set to TRUE, enables HTTP proxy authentication and implies that ENCRYPT=TRUE. This property is only applicable if `proxyHostname` is specified and `webSocketURL` is not specified.

Property	Value	Default	Description
`{ proxyPassword | proxy_password }`	`<password>`		Specifies the password for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the Cloud Connector location ID value. The value must be entered as a base64 encoded value with a "1." prepended to the string.
`{ proxyPort | proxy_port }`	`<port-number>`		Specifies the port of the proxy server. The default is 1080.
`{ proxyScpAccount | proxy_scp_account }`	SAP BTP, Cloud Foundry Environment: `<location-ID>` SAP BTP, Neo Environment: `<subaccount>.<location-ID>`		Provides SAP Cloud Connector routing information to the Cloud Connector SOCKS proxy. In an SAP BTP, Cloud Foundry Environment, provide an optional `<locationID>` in plaintext. Internally, the value is converted to base64 encoding. This value overrides the `proxyPassword` connection property. In an SAP BTP, Neo Environment, either `proxyScpAccount` or `proxyUsername` must be specified. `<subaccount>` is mandatory and `<locationID>` is optional. Both values are entered as plaintext strings and are internally converted to base64 encoding. This value overrides the `proxyUsername` connection property.

Property	Value	Default	Description
`{ proxyUserName | proxy_userid }`	`<user-name>`		Specifies the user name for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the JSON Web Token (JWT) value. When using an SAP HANA Cloud Connector in the SAP BTP, Neo Environment, this field is used to provide the subaccount and location ID details. The values must be base64 encoded manually using the format `1.base64(subaccount).base64(locationID)`.
`reconnect`	Boolean	TRUE	Controls whether the system automatically reconnects to the database instance after a command timeout or when the connection is broken. Reconnecting restores the old state (for example, if no transaction was open).

Property	Value	Default	Description
`resolveHostName`	IPv4First IPv6First	IPv4First	Controls which addresses to use in a server address, especially if an address or hostname resolves to multiple IP addresses. In all cases, stop attempting further resolved addresses once a TCP/IP connection is successfully established. The following values can be used with `resolveHostName`: • IPv4First (default) - try all resolved addresses starting with IPv4 addresses. • IPv6First - try all resolved addresses starting with IPv6 addresses.
`sessionVariable:<key-name>`	`<session variable value>`	None except for APPLICATION and APPLICATIONUSER keys	Sets the session variable `<key-name>` to `<session variable value>`. For example, to set the session variable with key `APPLICATION`, use `sessionVariable:APPLICATION=MyApplication`.

Property	Value	Default	Description
`statementCacheSize`	`<number>`	0 (no statements cached)	Controls the maximum number of cached prepared statements. Caching SQL statements that are frequently prepared and dropped can improve performance. DDL can cause behavior changes when statement caching is enabled. **Note** If it is possible that the definition of objects referred to by the prepared statements may change because DDL is executed while you are connected, then it is recommended that you do not enable prepared statement caching.
`statementRoutingFailureBackoff`	Boolean	TRUE	Enables or disables the re-establishment of an internal connection to a specific host for a time period, after the internal connection fails during the preparation or execution of a statement routed to that host. The minimum time period is 1 minute. The time period is doubled after each subsequent internal connection establishment or usage failure for the same node, up to a maximum of 1 hour. Every successful internal connection re-establishment and usage resets the time period to the minimum value.

Property	Value	Default	Description
`statementRoutingWarnings`	Boolean	FALSE	Returns a SQL warning to the application with SQLCODE -11001 or -11003 that gives information about the routing failure of a prepared statement.
`tcpKeepAliveCount`	`<number-of-retries>` This value is ignored if it is 0.	5	Controls the number of KeepAlive packets sent to the server before the driver assumes that the connection is closed. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.
`tcpKeepAliveIdle`	`<seconds>` This value is ignored if it is 0.	200	Controls the number of seconds that the connection is idle before the driver sends a KeepAlive packet. If 0 is specified, this value is determined by the operating system.
`tcpKeepAliveInterval`	`<seconds>` This value is ignored if it is 0.	4	Controls the number of specified seconds before the driver sends an additional KeepAlive packet if there was no response to the previous KeepAlive packet. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.

Property	Value	Default	Description
`timestampPadding`	Boolean	TRUE	In a timestamp where the format contains a fraction, this property determines whether the fractional portion of the timestamp will be 7 or 9 digits. When disabled, the fractional portion is 7 digits to match how the server displays timestamps. Otherwise, it is 9 digits. **Note** ABAP timestamps are not affected regardless of the value of `timestampPadding`.
`traceFile`			Specifies the trace file name. Only the stdout and stderr values are supported.
`traceOptions`			Implicitly enables tracing for all SQLDBC-based client connections. See the HDB_SQLDBC_TRACEOPTS description in the **Environment Variables** topic for more information.
`webSocketURL` (SAP HANA Service only)	`<webSocketUrl>`		Specifies the WebSocket URL used to connect to a SAP HANA server. The connection property value cannot be empty. WebSockets connections have the following default connection property settings: • encrypt=TRUE • compress=TRUE • reconnect=FALSE

Security Connection Properties

Property	Value	Default	Description
`{ cseKeyStorePassword | clientside_encryption_keystore_password }`	`<string>`		Provides the password for the local key store. This option is required when using client-side encryption. Passwords must be at least eight characters and have at least one upper case character, one lower case character, and one number.
`authenticationMethods`	`<comma-separated-string>`		Provides a comma-separated list of authentication methods to be used by the client. Possible values are: password, pbkdf2, ldap, saml, jwt, saplogon, kerberos, sessioncookie, and X509.
`authenticationX509`	`<string>`	`<value-of-sslKeyStore>`	Provides a string containing the X509 certificate or the name of a file containing the X509 certificate. The certificate must be PEM-encoded in PKCS8 format. Defaults to the existing value of `sslKeyStore`, if specified.
`authenticationX509Password`	`<password>`		With CommonCryptoLib, specifies the PIN for the PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`encrypt`	Boolean	FALSE	Enables or disables TLS encryption. For SAP HANA versions 2.5 and lower, set encrypt=TRUE when connecting to port 443 (SAP HANA Cloud). For SAP HANA versions 2.6 and higher, encrypt=TRUE is not required when connecting to port 443.

Property	Value	Default	Description
`sslCryptoProvider`	{ commoncrypto | openssl | mscrypto }	1. commoncrypto (if installed) 2. openssl/mscrypto	Specifies the cryptographic library provider used for TLS communication. If you specify a value for this property, then you must also explicitly specify paths in both the `sslKeyStore` and `sslTrustStore` properties to avoid configuration issues. If CommonCryptoLib is not available, OpenSSL is used by default in Linux environments, and msCrypto in Microsoft Windows environments. OpenSSL is not available on Microsoft Windows. **Note** Check the `client` folder of the installation package to see if `COMMONCRYPTOLIB.TGZ` is present. If not, you can download CommonCryptoLib separately. For instructions on how to download CommonCryptoLib, see "Download and Install SAP Common Crypto Library" in the *SAP HANA Client Installation and Update Guide*.
`sslHostNameInCertificate`	`<string>`	Empty	Specifies the host name used to verify server's identity. The host name specified here verifies the identity of the server instead of the host name with which the connection was established. For example, in a single-host system, if a connection is established from a client on the same host as the server, then a mismatch would arise between the host named in the certificate (actual host name) and the host used to establish the connection (localhost). If you specify ***** as the host name, then the server's host name is not validated. Other wildcards are not permitted.

Property	Value	Default	Description
`sslKeyStore`	`<file>` | `<PEM-encoded-identity>` msCrypto: MY | Root | Trust | CA	`$SECUDIR/sapcli.pse` (CommonCryptoLib) `$HOME/.ssl/key.pem` (OpenSSL) MY (msCrypto)	Specifies the path to the keystore file that contains the client's identity. An identity is used for mutual TLS authentication and consists of the client's private key, the client's certificate, and, optionally, the certificate of the signing authority that signed the client's certificate. If you are using CommonCryptoLib, then the PSE file must also contain the trust store (for example, the server's public certificates) and the sslTrustStore property should be empty. If you are using CommonCryptoLib, but not doing mutual TLS authentication, then the PSE file contains only the trust store. If you are using CommonCryptoLib, then use the SAPGENPSE tool (installed with CommonCryptoLib) to create the client PSE (`sapcli.pse`) and generate the client's keys. You must also import the server's public certificates into `sapcli.pse`. Typically, this is the root certificate or the certificate of the certification authority that signed the server's public certificates. See *SAP Note 1718944 (SAP HANA DB: Securing External SQL Communication (CommonCryptoLib))*. If you are using OpenSSL, use OpenSSL tools to create the required keystore file. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.

Property	Value	Default	Description
`sslMinProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	TLS12	Controls the minimum TLS version. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The lowest protocol version is accepted in TLS handshakes.
`sslMaxProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	MAX	Controls the maximum TLS version. MAX allows this parameter to be future-proof in the event that new TLS versions are supported. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The highest negotiated protocol version is used in TLS handshakes.
`sslKeyStorePassword`	`<password>`		With CommonCryptoLib, specifies the PIN for the keystore PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`sslSNIHostname`	`<string>`	Empty	Specifies the name of the host that is attempting to connect at the start of the TLS handshaking process.
`sslSNIRequest`	Boolean	TRUE	Specifies whether SNI requests are enabled for TLS connections: TRUE/FALSE.

Property	Value	Default	Description
`sslTrustStore`	`<file>` | `<PEM-encoded-trust-store>` msCrypto: MY | Root | Trust | CA	`$HOME/.ssl/trust.pem` (OpenSSL) sapcli.pse (CommonCryptoLib) MY (msCrypto)	Specifies the path to a trust store file that contains the server's public certificates if using OpenSSL. Typically, the trust store contains the root certificate or the certificate of the certification authority that signed the server's public certificates. If you are using the cryptographic library CommonCryptoLib or msCrypto, leave this property empty. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.
`sslUseDefaultTrustStore`	Boolean	TRUE	(Only applies to the OpenSSL Crypto Provider) Set to FALSE in order to disable the use of the default Certificate Authorities provided with OpenSSL or the Operating System.
`sslValidateCertificate`	Boolean	TRUE	Specifies whether to validate the server's certificate.

4.2.5 Driver-Specific Extensions

The golang driver supports certain extensions to the SQL package API to allow for additional HANA-specific features.

Note that these features may not be portable to other golang driver implementations.

4.2.5.1 LOB Streaming for INSERT, UPDATE, DELETE, and SELECT Statements

The Go driver supports streaming data to/from LOB columns in chunks, using driver-specific LOB objects as parameters.

Context

The Go driver supports streaming data to/from LOB columns by providing LOB streaming objects to statements that are prepared from a sql.Db* or sql.Tx* object. This allows for the following when sending data to or retrieving data from LOB columns:

- More control over the driver's internal memory usage
- Greater flexibility in sources of data for input/output other than directly in the application's memory

> **Note**
>
> This feature only works with golang runtime 1.9.x or later.

Procedure

1. Import the driver into an accessible alias name:

   ```
   Import driver "SAP/go-hdb/driver"
   ```

2. Use the Go `lobChunkSize` connection property to specify the chunk size of the data that is retrieved and sent by the driver.

   ```
   db, err = db.Open("<basic-connection-string>?<key-value-pairs-for-attributes>; lobChunkSize=<size>
   ```

   > **Note**
   >
   > Points to note:
   >
   > - The default `lobChunkSize` (if unspecified in the DSN string) is 1048576 bytes.
   > - The `<size>` value must be between 8192 bytes and 2147483647 bytes, otherwise an error is returned by the operation that establishes the connection. The more specific error can be checked using the detailed Ping Error feature after creating a sql.DB object via db.Open().

3. For INSERT, UPDATE, and DELETE statements:
   1. Prepare the INSERT, UPDATE, or DELETE statement:
      1. On the sql.DB object directly

         ```
         stmt, err = db.Prepare("<INSERT/UPDATE/DELETE-statement>")
         // ...
         ```

2. In a transaction

```
tx, err = db.Begin()
// ...
stmt, err = tx.Prepare("<INSERT/UPDATE/DELETE-statement>")
// ... (The code here is described in the next step)
// tx.Commit() or tx.Rollback()
```

2. Provide the data to be inserted into the LOB column using an io.ReadCloser object.
   1. If the io.ReadCloser object wraps an object using the Operating System systemcall feature to read and chunk the data (for example, a file), then it can be provided to stmt.Exec() directly in the main thread:

```
import OS // Add to the top with the rest of the imports
// ...
file, err = os.Open("<file-name>")
// ...
lob := driver.newWritelob(file)
res, err = stmt.Exec(lob)
```

   2. If the memory/data is coming from the application directly, then an io.PipeReader/io.PipeWriter pair must be used and the io.PipeReader must be provided to stmt.Exec() in a go routine:

```
pipeReader, pipeWriter := io.Pipe()
lob = driver.NewWriteLOB(pipeReader)
//create some []byte data and store in a variable "data"
// Create a go-routine to do the insert
var err error
errors := make(chan error)
go func() {
    res, err := insertStmt.Exec(lob)
    if err != nil {
        errors <- err
    }
    errors <- nil
}()
// You must put the data to write in the main thread
pipeWriter.Write(data)
pipeWriter.Close()
err := <-errors
// ...
```

3. The driver processes the WriteLOBs in the order they are provided to stmt.Exec(). If any errors occured during processing of a WriteLOB input parameter, stmt.Exec() will:
   1. Close any subsequent WriteLOBs without processing them.
   2. Return an error indicating the specific WriteLOB processing that resulted in the error.
   3. No data will be sent to the server and the insert will not be executed.

4. For SELECT statements:
   1. Prepare the SELECT statement.
   2. When executing the query, a driver-specific context object must be provided to all Query functions (QueryContext()/QueryRowContext()) to indicate that all fetches from LOB columns will be done using a ReadLOB type:

```
import context // Add to the top with the rest of the imports
// ...
ctx := driver.NewReadLobFetchContext(context.Background())
```

3. Use an io.WriteCloser to fetch the data returned from the LOB column:
    1. If the io.WriteCloser object wraps an object using the Operating System systemcall features to read/chunk the data (e.g. a file), then provid it to stmt.Exec() directly in the main thread:

```
import os // Add to the top with the rest of the imports
//...
file, err = os.Open("<file-name>")
// ...
// If the data is guaranteed not to be NULL, use a ReadLOB
(driver.NewReadLOB(<file>))
lob := driver.NewNullReadLOB(<file>)
res, err = stmt.Exec(lob)
// Assuming the query returns one row here:
stmt.QueryRowContext(ctx).Scan(lob)
// Check if the data is NULL
if lob.Valid {
} else {
// Non-NULL data
}
```

    2. If the memory/data is being used by the application directly, then an io.PipeReader/io.PipeWriter pair must be used, and the io.PipeReader must be provided to stmt.Exec() in a go-routine:

```
var dataBuffer bytes.Buffer
pipeReader, pipeWriter := io.Pipe()
// Execute the query in a separate go-routine
var errorFromStmt error
errors := make(chan error)
go func() {
    ctx := NewReadLobFetchContext(context.Background())
    // Assuming the query returns one row.
    // This would be a general error from the query execution or scan
    err := stmt.QueryRowContext(ctx, intVal).Scan(pipeWriter)
    errors <- err
}()
// Actually read the data in the main thread
_, err := dataBuffer.ReadFrom(pipeReader)
pipeReader.Close()
errFromStmt = <- errors
```

> ### Note
>
> - The driver processes the LOBs in order they are provided to the rows.Scan() call. If any errors occurred during the processing of a ReadLOB and the io.ReadCloser is a io.PipeWriter, then the error is returned directly by any Read call on the io.PipeReader.
>   Due to the way in which the rows.Scan() is implemented, readers are not tracked internally. It is the responsibility of the application to check for an error on a Read call on io.PipeReader and react accordingly (especially, not trying to read from subsequent pipeReaders, but instead just calling Close() on them to release resources).
> - driver.NullReadLOB implements the Valuer interface (the standard for most sql.Null* types), and it returns the io.WriteCloser that was provided. However, its return value is not expected to be used outside of verifications (it is already closed after the rows.Scan() call).

4.2.5.2 Obtaining Specific Ping() Errors

The GO Driver supports the retrieval of SAP HANA protocol-specific error messages sql.DB.PingContext() and sql.Conn.PingContext() by providing a driver-specific context.Context object.

Context

By default, the GO Driver returns driver.ErrBadConn when a call to sql.DB.PingContext() or sql.Conn.PingContext() fails. However, applications may want more detailed database error messages in order to analyze a potential problem with the connection. To do this, the Go driver allows you to request more detailed error messages by providing a driver-specific context object to sql.DB.PingContext() and sql.Conn.PingContext() calls.

Procedure

1. Import the driver into an accessible alias in the list of imports:

   ```
   import driver "SAP/go-hdb/driver"
   ```

2. In the application code, after opening a new hana sql.DB connection, create a new context object using the module function, NewPingDetailedErrorContext():

   ```
   ctx := driver.NewPingDetailedErrorContext()
   ```

3. Pass this context object to the sql.DB.PingContext()/sql.Conn.PingContext() call to get the SAP HANA-specific error if the call fails:

   ```
   err := db.PingContext(ctx)
   // Check the error message
   ```

4.2.5.3 Perform Bulk INSERT, UPDATE, and DELETE Statements

The Go driver supports bulk INSERT, UPDATE, and DELETE statements that have been prepared from a `sql.Db*` or `sql.Tx*` object.

Context

Bulk INSERT, UPDATE, and DELETE statements with LOB streaming are not supported.

golang internal processing depends on the dbcapiHDB C library for its implementation, so it tries to batch as many arrays together as possible, up to the array size, based on the types of equal values in consecutive array

cases. If types differ in any of the arrays relevant to the first array considered for the batch, the current batch of similar values is sent off first. The execution then continues batching any remaining arrays until all row arrays are inserted. As a result, it is recommended to use the same types for the array values. Note that NULL always matches the previous type considered.

> **Note**
>
> This feature only works with golang runtime 1.9.X or newer. A batch operation returns an execution error when used with an earlier golang runtime version.

To execute an insert, update, or delete with a batch of parameters, complete the following steps:

Procedure

1. Prepare the insert/update/delete statement.

   ```
   stmt, err := db.Prepare("<INSERT/UPDATE/DELETE statement>")
   ```

2. Create the batch of row values for the insert/update/delete.

   Each row is an array of interface objects (`[]interface{}`) and the batch is an array of these rows (`[][]interface{}`).

   The following rules must be followed when creating the batch of rows:

   - Create the batch of rows (`type [][]interface{}`) with each `[]interface{}` array containing the values for each column of that row. A `nil` value is interpreted as a NULL for the column.
   - Each row array provided in the batch of values must be the same length, and at least as long as the number of input parameters expected.

   The following input parameter types are supported as row array values for bulk inserts:

   - int8, int16, int32, int64
   - float32, float64
   - bool
   - []byte
   - String
   - time.Time
   - All `sql.Null*` types

   The following example inserts the values into all columns of a table defined with a column sequence of `A INTEGER, B DOUBLE`:

   ```
   batchSize := 20
   data := make([][]interface{}, batchSize)
   len
   for i := 0; i < batchSize; i++ {
       data[i] = []interface{}{int16(i), float64(i)}
   }
   ```

3. Provide the created data array as a single input parameter to `stmt.Exec()`.

   ```
   res, err := stmt.Exec(batchOfRows)
   ```

4.2.5.4 Perform Bulk Fetching

The Go driver supports bulk fetching.

Context

The following table shows the combination of bulk and row array value types, and what column types can be fetched:

Bulk Type	Row Array Value Type	Column Type
`BulkNullInt64`	int64	TINYINT, SMALLINT, INT[EGER], BIGINT
`BulkNullFloat64`	float64	REAL, DOUBLE
`BulkNullBool`	bool	BOOLEAN
`BulkNullTime`	time	DATE, TIME, TIMESTAMP
`BulkNullByte`	byte	BINARY, VARBINARY
`BulkNullString`	String	[N]CHAR, [N]VARCHAR, DECIMAL[(p[,s])], all other DECFLOAT/DECIMAL aliases

> **Note**
>
> All other special types/LOB columns are not supported through the bulk fetch interface.
>
> This feature only works with golang runtime 1.9.X or newer. A batch operation returns an execution error when used with an earlier golang runtime version.

To bulk fetch rows, complete the following steps:

Procedure

1. Import the driver into an accessible alias name:

```
import driver "SAP/go-hdb/driver"
```

2. Create a new context to represent the fetch size, using the package function NewFetchSizeContext().

```
ctx, err := driver.NewFetchSizeContext(context.Background(),
uint(<rowsetFetchSize>))
```

3. When fetching from a rows object, use a custom `BulkNull` type from the package with the function `rows.Scan()`.

```
var values <bulk-type>

for rows.Next() {
   err = rows.Scan(&value)

   if err != nil {
     // handle error
  }

  for  i := 0; I < values.NumberOfRows; i++ {
       if values.Valids[i] { // Not Null
             value := values.<type-array-name>[i]
             // Use value as required
       } else {
           // values.<type-array-name>[i] is its underlying type's zero-
value, and the actual value fetched from the database is NULL
      }
}
```

4.2.5.5 Vendor-Specific Errors Returned from SQL Package API Calls

The golang driver supports getting more detailed error objects from all SQL package API functions which can return an error.

Context

The golang sql package API contains many functions which can return errors to the application. The golang driver provides a detailed error object through these API functions, when errors/warnings occur that give the following information:

- **Access to warnings**
  - This must be explicitly enabled by providing the golang-only connection property - golangReportWarningsWithErrors=true (default = false)
- **Access to a collection of errors/warnings**
  - SAP HANA can return multiple errors and/or warnings for a single API call that has failed.
- **The following information can be obtained from the error object:**
  - - A string representation of the error collection, using Error(), which maintains the original error output from previous versions of the driver for backwards compatibility, or String(), which provides information representing the entire set of errors/warnings in the collection
    - Access to all errors/warnings that have occurred as a result of the previous sql package API call
    - Access to the count of errors + warnings (an uint, using NumberOfErrors())
    - Going to the next error, using NextError(), which returns a bool, indicating whether or not there was another error in the collection
    - Going to a specific error, using SetErrorPosition(index uint), and returns and error which describe whether or not the index was successfully set

- Resetting the index to the beginning of the collection (via Reset)
- For a specific error/warning in the collection, it gives access to:
  - The Error Code (an int, using ErrorCode())
  - The Error Text (a string using ErrorText())
  - The SqlState (a string, using SqlState())
  - The ErrorLevel (a driver-specific enum constant defined as driver.HdbServerityLevel, using ErrorLevel())

    > **Note**
    >
    > HdbSeverityLevel has the following levels:
    >
    > - driver.ErrorLevel_Warning
    > - driver.ErrorLevel_Error
    > - driver.ErrorLevel_Faal

  - Checker functions for the Severity Level (IsWarning() , IsError())
  - The Row Number (a 1-based index of which row the error/warning applies to. This only applies to batch INSERT/UPDATE/DELETE operations, otherwise the value is zero)

Procedure

To gain access to the detailed error message, complete the following steps:

1. Import the driver package into a named alias.

```
import driver "SAP/go-hdb/driver"
```

2. Get the error from the SQL package API function call.

```
...
_, err = db.Exec("Invalid")
```

3. Get the more detailed error from the one returned in order to access the information.

```
v := reflect.ValueOf(err)
hdbError, ok := v.Interface().(*driver.HdbError)
numberOfErrors := hdbError.NumberOfErrors()
errorLevel := hdbError.ErrorLevel() ...
errorCode := hdbError.ErrorCode()
errorText := hdbError.ErrorText()
sqlState := hdbError.SqlState()
isError := hdbError.IsError()
isWarning := hdbError.IsWarning()
err = hdbError.SetErrorPosition(0)
// Use the error information
hdbErr.Reset()
hasNextError := hdbErr.NextError() // Go to the next error if there is any
```

4.2.6 Uninstall the Go Driver

Remove the Go driver files from your computer.

Procedure

1. Delete the source directory, located in `$GOPATH/src`.
2. Delete the compiled package file that is located in `$GOPATH/pkg/<architecture>`.

   For example, `%GOPATH%\pkg\windows_amd64`.

4.3 JDBC Application Programming

Java Database Connectivity (JDBC) is a Java-based application programming interface (API) which includes a set of functions that enable Java applications to access a data model in a database. The SAP HANA client includes a dedicated JDBC interface.

The JDBC driver is available with the SAP HANA client install and from the central Maven repository under the terms of the SAP Developer License Agreement.

> **Note**
>
> To find the HANA JDBC driver on Maven, search *com.sap.cloud.db.jdbc*.

Related Information

SAP Developer License Agreement
Central Maven Repository

4.3.1 Connect to SAP HANA via JDBC

SAP HANA provides a driver that enables Java applications to connect to the SAP HANA database with the JDBC application programming interface (API).

Procedure

1. Install the JDBC driver.

   The driver (`ngdbc.jar`) is installed as part of the SAP HANA client installation and by default is located at:
   - `C:\Program Files\sap\hdbclient\` on Microsoft Windows platforms
   - `/usr/sap/hdbclient/` on Linux and UNIX platforms
2. Add `ngdbc.jar` to your classpath.
3. Write Java code to create a connection to the database and execute SQL commands. Use a connection string in the form of `jdbc:sap://<server>:<port>[/?<options>]`. For example:

   ```
   jdbc:sap://myServer:30015/?autocommit=false
   ```

   Specify one or more failover servers by adding additional hosts, as in the following example:

   ```
   jdbc:sap://myServer:30015;failover1:30015;failover2:30015/?autocommit=false
   ```

   To connect to a specific database, for example `tdb1`, use the `databaseName` parameter, as illustrated in the following code:

   ```
   jdbc:sap://localhost:30013/?databaseName=tdb1&user=SYSTEM&password=manager
   ```

Example

The following is an example of connecting to an SAP HANA server called `myhdb`, which was installed as instance `07`, with user name `myName` and password `mySecret`. Make sure to change these for your system, and add the JDBC driver (ngdbc.jar) to your classpath.

```
import java.sql.*;
public class jdemo {
   public static void main(String[] argv) {
      Connection connection = null;
      try {
         connection = DriverManager.getConnection(
            "jdbc:sap://myhdb:30715/?autocommit=false", "myName",
               "mySecret");
      } catch (SQLException e) {
         System.err.println("Connection Failed:");
         System.err.println(e);
         return;
      }
      if (connection != null) {
         try {
            System.out.println("Connection to HANA successful!");
            Statement stmt = connection.createStatement();
```

```
            ResultSet resultSet = stmt.executeQuery("Select
               'Hello, world' from dummy");
            resultSet.next();
            String hello = resultSet.getString(1);
            System.out.println(hello);
       } catch (SQLException e) {
          System.err.println("Query failed!");
       }
     }
   }
}
```

Related Information

Connecting to SAP HANA Databases and Servers [page 8]

4.3.1.1 JDBC Command-Line Connection Options

A number of parameters enable you to test the connection to a database by running a simple but specific query.

The `ngdbc.jar` file enables you execute simple commands or check if a connection to the SAP HANA database can be established with the JDBC client.

```
java -jar ngdbc.jar -u <user,password>
                    [-n <hostname:port>]
                    [-i <instance-number>]
                    [-d <database-name>]
                    [-o <connect-option>]
                    [-w <maximum-column-width>
                    [-c <sql-command>]
```

Connections with an Instance Number and Database Name

The `-i` and `-d` options specify the instance number and database name (for example, `tdb1`) to use for the connection to SAP HANA:

```
java -jar ngdbc.jar -u MyUser,MyPassword1 -i 00 -n myServer -d tdb1 -c "SELECT
DATABASE_NAME FROM SYS.M_DATABASES"
```

Connections with SYSTEMDB Port and Database Name

The `-n` option specifies the port number to use for connections to the SAP HANA database; the `-d` option enables you to provide the name of a specific database instance (for example, `tdb1`):

```
java -jar ngdbc.jar -u MyUser,MyPassword1 -n myServer:30013 -d tdb1 -c "SELECT
DATABASE_NAME FROM SYS.M_DATABASES"
```

Connections with the SYSTEMDB Port and Database Name as a Property

The `-o` parameter specifies the name of a specific database instance (for example, `tdb1`) as a connection option:

```
java -jar ngdbc.jar -u MyUser,MyPassword1 -n myServer:30013 -o databaseName=tdb1
-c "SELECT DATABASE_NAME FROM SYS.M_DATABASES"
```

Connections with a SQL Command

Use the `-c` parameter to test a SQL query. The driver chooses reasonable defaults for the display column widths for variable length columns, for example, 200 characters for LOB, spatial, and array columns, and a maximum of 100 characters for character and binary columns. Using the `-w <maximum-column-width>` option with this parameter allows you to override these defaults.

> **Note**
>
> The specified value will only be used for all variable length columns (integer, floating-point, and date/time columns are not affected).

The following example limits the display width for all variable length columns to 50 characters:

```
java -jar ngdbc.jar … -w 50 -c <sql-command>
```

4.3.1.2 JDBC Connection Options in Java Code

It is possible to establish a connection to the SAP HANA database directly from the Java code you write.

The JDBC database connection URL is extended to include the database name and the instance number as properties. With `ngdbc.jar` set in the environment variable `<CLASSPATH>`, the method to establish a connection to the SAP HANA database is `java.sql.DriverManager.getConnection()`.

java.sql.DriverManager.getConnection(url, user, password)

Use the `java.sql.DriverManager.getConnection` method to connect to SAP HANA with a URL, a user name, and the corresponding password.

```
import java.sql.Connection;
import java.sql.DriverManager;
java.sql.Connection conn = java.sql.DriverManager.getConnection
       ("jdbc:sap://localhost:30013/?databaseName=tdb1", "SYSTEM",
        "manager");
```

Use the `databaseName=<DB_Name>` option to extend the URL to specify the name of a particular SAP HANA instance to connect to, for example.

```
import java.sql.Connection;
import java.sql.DriverManager;
java.sql.Connection conn = java.sql.DriverManager.getConnection
       ("jdbc:sap://localhost/?instanceNumber=00&databaseName=tdb1",
        "SYSTEM", "manager");
```

java.sql.DriverManager.getConnection(url)

Use the `java.sql.DriverManager.getConnection` method to connect to SAP HANA with a URL. The URL can be extended to include the connection options; for example: the user name and corresponding password, and the database name.

```
import java.sql.Connection;
import java.sql.DriverManager;
java.sql.Connection conn =
java.sql.DriverManager.getConnection
("jdbc:sap://localhost:30013/?databaseName=tdb1&user=SYSTEM&password=manager");
```

java.sql.DriverManager.getConnection(url, properties)

Use the `java.sql.DriverManager.getConnection` method to connect to SAP HANA with a URL, which you then extend using information specified in properties, as illustrated in the following example.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
java.util.Properties info = new java.util.Properties();
info.put("databaseName", "tdb1");
info.put("user", "SYSTEM");
info.put("password", "manager");
java.sql.Connection conn = java.sql.DriverManager.getConnection
                         ("jdbc:sap://localhost:30013", info);
```

4.3.1.3 JDBC Connection Properties

When connecting to an SAP HANA database using JDBC, there are several connection properties that you can specify.

The following table lists the JDBC connection properties, which are case insensitive, that are specific to SAP HANA clients:

Property	Value	Default	Description
`allowLocalCompress`	Boolean	TRUE	Determines whether or not connections from the same machine as the SAP HANA server are compressed. The `compress` parameter and/or the `indexserver.ini` > *session* > *compression* server configuration control whether or not network compression is enabled. **Note** If the connection uses a proxy, do not use `allowLocalCompress`=FALSE as the connection may not be compressed when required.
`autocommit`	Boolean	TRUE	Controls whether statements are automatically committed. Specify TRUE for autocommit mode and FALSE for commits and/or rollbacks to be performed manually.
`bindAddress`	`<ip-address>`	NULL	Specifies that the provided address is used as the source IP/host for the socket connection. By default, the address preferred by the OS is used.

Property	Value	Default	Description
`closeHandlesByCleaner`	Boolean	TRUE when running JDK 9 and later. Otherwise, this is specified as FALSE. **Note** This property is not supported for JDK 8 and earlier.	When enabled, connections, statements, and result sets are automatically closed when they become phantom-reachable.
`closeHandlesOnFinalize`	Boolean	Specified as TRUE when running JDK 8 and earlier. Otherwise, specified as FALSE.	When enabled, connections, statements, and result sets automatically close when their Java finalizers are run.
`communicationTimeout`	`<number>` (milliseconds)	0 (disabled)	Aborts communication after the specified timeout. Setting this option to 0 disables the timeout.
`compress`	Boolean	FALSE	Enables or disables network compression for the connection. The `indexserver.ini` > *session* > *compression* server configuration defaults to FALSE. Setting this parameter overrides the setting of the `indexserver.ini` > *session* > *compression* parameter. The `allowLocalCompress` property controls if clients running on the same machine as the server use network compression when compression is requested.

Property	Value	Default	Description
`connectTimeout`	`<number>` (seconds)	0 (disabled) for versions 2.4 and lower 60 seconds for versions 2.5 and higher	Aborts the connection attempt after the specified maximum timeout. The connection attempt typically fails immediately if the server cannot be found. **Note** This connection property corresponds to SQLDBC's `nodeConnectTimeout` connection property, not its `connectTimeout` connection property. There is no JDBC-equivalent to SQLDBC's `connectTimeout` connection property.
`currentSchema`	`<schema>`	CURRENT_USER	Sets the current schema which is used for identifiers without schema.

Property	Value	Default	Description
`cursorHoldabilityType`	NONE, COMMIT, ROLLBACK, COMMIT_AND_ROLLBACK	COMMIT	Specifies whether the cursor for a result set remains open after the transaction is committed or rolled back. If a cursor remains open, then the application is able to fetch more rows from it. NONE means the cursor is always closed after a commit or rollback, while COMMIT means the cursor remains valid after a commit, but not after rollback. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources. Support for holding a cursor over ROLLBACK or COMMIT_AND_ROLLBACK is available in SAP HANA servers beginning with SAP HANA 2.4.
`databaseName` (SAP HANA Platform only)	`<name>`	Empty	Specifies the name of the database to connect to. Attempts to use this property to connect to an SAP HANA Cloud database could cause the connection to fail.

Property	Value	Default	Description
`deferredPrepare`	Boolean	FALSE	Reduces the number of round trips between the client and the SAP HANA server by not sending a Prepare request to the server when Connection.prepareStatement() or Connection.prepareCall() is invoked. Enabling this property helps improve performance for applications by reducing the number of client-server round-trips without changing application logic. Statement routing must be disabled or you cannot use deferred prepares. If the deferredPrepare connection property is set to true: • Client-side encryption is not supported. • Character-to-binary and binary-to-character conversions for input arguments are not supported. • The PreparedStatement/CallableStatement methods are not supported for the following SAP HANA data types: • setString(), setNString(), setCharacterStream(), setNCharacterStream(), setAsciiStream(), setClob(), and setNClob() for BINARY, VARBINARY, BLOB, and BINTEXT parameters.

Property	Value	Default	Description
			• setBytes(), setBinaryStream(), and setBlob() for CHAR, VARCHAR, NCHAR, NVARCHAR, SHORTTEXT, CLOB, NCLOB, TEXT, and BINTEXT parameters.
`distribution` (SAP HANA Platform only)	OFF, CONNECTION, STATEMENT, ALL	STATEMENT	Specifies the distribution mode. Specifying STATEMENT does not include CONNECTION distribution.

Property	Value	Default	Description
`emptyTimestampIsNull`	Boolean	TRUE	When enabled, inserts DATE, TIME, SECONDDATE, and TIMESTAMP values as empty strings that are returned as NULLs. When disabled, these values are returned as the following out-of-band values: • `DATE` - one day before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 00:00.000000000 UTC BC • `TIME` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `SECONDDATE` - one second before, 0001-01-01 00:00:00.0000000 UTC AD which is 0001-12-31 23:59:59.000000000 UTC BC • `TIMESTAMP` - one hundred nanoseconds before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.999999900 UTC BC To compare a java.sql.Date/Time/Timestamp value against the relevant out-of-band value, use the following values: • For one day before 0001-01-01

Property	Value	Default	Description

			00:00:00.0000000 (-62135769600000 - 86400000 = -62135856000000), use new Date(-6213585600000 0)
			• For one second before 0001-01-01 00:00:00.0000000 (-62135769600000 - 1000 = -62135769601000), use new Time(-6213576960100 0)
			• For one hundred nano-seconds before 0001-01-01 00:00:00.0000000 (-62135769600000 - 1 = -62135769600001), use Timestamp t = new Time-stamp(-621357696000 01); // t is one millisec-ond before 0001-01-01 00:00:00.0000000 t.setNa-nos(999999900); // now t is one hundred nanoseconds before 0001-01-01 00:00:00.0000000 Alternatively, with the SAP HANA 2.0 SPS 01 and later JDBC driver, you can use the following static meth-ods defined in the com.sap.db.jdbc.Driver class:
			public static boolean isEmpty-Date(java.sql.Date date)

Property	Value	Default	Description
			public static boolean isEmptyTime(java.sql.Time time) public static boolean isEmptyTimestamp(java.sql.Timestamp timestamp) These methods return TRUE if the specified java.sql.Date/Time/Timestamp is one of the above out-of-band values; otherwise, they return FALSE.
`ignoreTopology` (SAP HANA Platform only)	Boolean	FALSE for SAP HANA client versions 2.4 and lower TRUE for SAP HANA client versions 2.5 and greater	Controls how the client uses the topology returned from the server: TRUE = Use the topology unless port-forwarding is detected FALSE = Always ignore the topology It is an error to specify both ignoreTopology=TRUE and siteType=PRIMARY or SECONDARY.

Property	Value	Default	Description
`inexactDoubleToDecimalConversion`	Boolean	FALSE for SAP HANA client versions 2.4 and lower TRUE for SAP HANA client versions 2.5 and greater	Controls the behavior of PreparedStatement.setFloat/Double() for DECIMAL parameters: **FALSE** The intermediate java.math.BigDecimal value is created from the specified float/double value using `new BigDecimal( value )`. The BigDecimal value is an exact representation of float/double binary floating-point value. **TRUE** The intermediate java.math.BigDecimal value is created from the specified float/double value using `BigDecimal.valueOf( value )`. The BigDecimal value is obtained from the canonical string representation of the float/double binary floating-point value. This behavior provides compatibility with Oracle databases.
`instanceNumber`	`<number>`	Empty	Specifies the SAP HANA server instance number. If a server port is specified in the connection URL, then the instanceNumber property is ignored.
`isolation`	TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE	TRANSACTION_READ_COMMITTED	Sets the isolation level for the connection.
`key`	`<key>`	Empty	Specifies the key for the hdbuserstore.
`locale`	ISO locale code	client locale	Specifies the client locale.

Property	Value	Default	Description
`maxLazyDroppedStatements`	`<number>`	50	Controls the maximum number of queued statements to be dropped. Setting this value to 0 disables the lazy drop feature and causes a request to be sent to the server each time a statement is dropped by the application.

Property	Value	Default	Description
`networkGroup` (SAP HANA Platform only)	`<customMappingName>`	Value from the HDB_NETWORK_GROUP environment variable; otherwise, use public host name mappings	Allows clients running in different networks that need to use different addresses to reach a particular SAP HANA node to specify the custom host name mappings required. This property is used in cases where the public host name addresses defined by the public_hostname_resolution server configuration cannot be reached by the client and you want to use statement routing, connection distribution, and/or active/active read-enabled hint routed to secondary connections. In these cases, you must first define custom host name mappings by using the `public_hostname_resolution_<customMappingName>` server configuration. Then you can specify this `<customMappingName>` with either with the HDB_NETWORK_GROUP environment variable or the networkGroup connection property. `<customMappingName>` is case insensitive and must contain only 7-bit ASCII characters. For more information, see "Custom Mapping Host Names for Database Client Access" in the *SAP HANA Administration Guide*.

Property	Value	Default	Description
`nonBlockingIO`	Boolean	TRUE	Controls whether non-blocking I/O is used for sending and receiving packets. The original I/O behaviour can be re-enabled by setting the nonBlockingIO connection property to false.
`packetSize`	`<number>` bytes	130,000 bytes SAP HANA client versions 2.4 and lower 1 MB for SAP HANA client versions 2.5 and greater	Sets the maximum size of a request packet sent from the client to the server in bytes. The minimum is 130, 000 bytes.
`password`	`<password>`	Empty	Specifies the user password. Optional, depending on the authentication method used.
`prefetch`	Boolean	TRUE	Controls result set prefetching. When enabled, result set prefetch requests the next block of rows of a result set while the application is still processing the current block of rows. When fetching large result sets, result set prefetch can improve performance; however, it can also result in the client using more memory. In some cases, the server JobExecutor (SAP HANA Cloud) thread could be blocked from sending the reply for an indefinite period of time until the client receives the prefetch reply. If the result set contains LOB columns, then result set prefetch is always disabled.

Property	Value	Default	Description
`proxyHostname`	`<host-name>`	Empty	Specifies the host name of the SOCKS5 proxy server for TCP/TLS connections or an HTTP proxy server for TCP/TLS/WebSocket connections. See `proxyHttp` on how to choose between SOCKS and HTTP proxy authentication.
`proxyHttp`	Boolean	FALSE	Enables SOCKS proxy authentication when not specified or set to FALSE. When set to TRUE, enables HTTP proxy authentication and implies that ENCRYPT=TRUE. This property is only applicable if `proxyHostname` is specified and `webSocketURL` is not specified.
`proxyPassword`	`<password>`	Empty	Specifies the password for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the Cloud Connector location ID value. The value must be entered as a base64 encoded value. See `proxyHttp` on how to choose between SOCKS and HTTP proxy authentication.

Property	Value	Default	Description
`proxyPort`	`<port-number>`	Empty	Specifies the port of the SOCKS5 proxy server or HTTP proxy server. The default is 1080 for TCP connections and 80 for WebSocket connections. See `proxyHttp` on how to choose between SOCKS and HTTP proxy authentication.
`proxyScpAccount`	SAP BTP, Cloud Foundry Environment: `<location-ID>` SAP BTP, Neo Environment: `<subaccount>.<location-ID>`	Empty	Provides SAP Cloud Connector routing information to the Cloud Connector SOCKS proxy. In an SAP BTP, Cloud Foundry Environment, provide an optional `<locationID>` in plaintext. Internally, the value is converted to base64 encoding. This value overrides the `proxyPassword` connection property. In an SAP BTP, Neo Environment, either `proxyScpAccount` or `proxyUsername` must be specified. `<subaccount>` is mandatory and `<locationID>` is optional. Both values are entered as plaintext strings and are internally converted to base64 encoding. This value overrides the `proxyUsername` connection property. See `proxyHttp` on how to choose between SOCKS and HTTP proxy authentication.

Property	Value	Default	Description
`proxyUserName`	`<user-name>`	Empty	Specifies the user name for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the JSON Web Token (JWT) value. When using an SAP HANA Cloud Connector in the SAP BTP, Neo Environment, this field is used to provide the subaccount and location ID details. The values must be base64 encoded manually using the format `1.base64(subaccount).base64(locationID)`. See `proxyHttp` on how to choose between SOCKS and HTTP proxy authentication.
`readOnly`	Boolean	FALSE	Specifies whether read-only statements or all statements are permitted. Attempting to execute DDL or DML causes an exception.
`reconnect`	Boolean	TRUE	Controls whether the system automatically reconnects to the database instance after a command timeout or when the connection is broken. Reconnecting restores the old state (for example, if no transaction was open).

Property	Value	Default	Description
`respectAutoCommitOnPrepare`	Boolean	FALSE	Unblocks server garbage collection if `autocommit` is set to TRUE and a call to Connection.prepareStatement() or prepareCall() throws a SQLException. **Note** Specifying `respectAutoCommitOnPrepare` to TRUE unblocks server garbage collection on a failed prepare, but also closes any result sets opened with CLOSE_CURSORS_ON_COMMIT holdability.

Property	Value	Default	Description
`routeDirectExecute` (SAP HANA Platform only)	Boolean	FALSE	Converts direct execute into prepare and execute (routed execute) if the number of index servers is more than one and if statement routing is enabled. This property increases performance only if the gains from statement routing are more than the cost of the additional round trips required. Otherwise, this property can result in reduced performance. For example, if you are running a query on tables that are located on nodes other than the node you are currently connected to. To improve the performance of this property, enable prepared statement caching with the statementCacheSize connection property. **Note** This property applies to single direct executes only. Batches of direct executes are never routed.
`sessionVariable:<key-name>`	`<session-variable-value>`	None except for APPLICATION and APPLICATIONUSER keys	Sets the session variable `<key-name>` to `<session variable value>`. For example, to set the session variable with key `APPLICATION`, use `SESSIONVARIABLE:APPLICATION=MyApplication`.

Property	Value	Default	Description
`siteType` (SAP HANA on-premise only)	PRIMARY, SECONDARY	None	Specifies whether the connection is made to either the PRIMARY or SECONDARY site in an Active/Active (read enabled) system, skipping any of the `<host>:<port>` locations that are for a different site type. You must set this property for takeover/failover situations that use connection distribution or non-virtual IP addresses. It is an error to set both ignoreTopology and siteType.
`splitBatchCommands` (SAP HANA Cloud only)	Boolean	FALSE	Allows split and parallel execution of batch commands on partitioned tables.
`statementCacheSize`	`<number>`	0 (no statements cached)	Controls the maximum number of cached prepared statements. Caching SQL statements that are frequently prepared and dropped can improve performance. DDL can cause behavior changes when statement caching is enabled. **Note** If there is a possibility that the definition of objects referred to by the prepared statements may change due to DDL while you are connected, then it is recommended that prepared statement caching is disabled.

Property	Value	Default	Description
`statementRoutingFailureBackoff`	Boolean	TRUE	Enables or disables the re-establishment of an internal connection to a specific host for a time period, after the internal connection fails during the preparation or execution of a statement routed to that host. The minimum time period is 1 minute. The time period is doubled after each subsequent internal connection establishment or usage failure for the same node, up to a maximum of 1 hour. Every successful internal connection re-establishment and usage resets the time period to the minimum value.
`tcpKeepAliveCount`	`<number-of-retries>` This value is ignored if it is 0.	5	Controls the number of KeepAlive packets sent to the server before the driver assumes that the connection is closed. This property applies to Linux and MacOS only. On Microsoft Windows this value is determined by the operating system.
`tcpKeepAliveIdle`	`<seconds>` This value is ignored if it is 0.	200	Controls the number of seconds that the connection is idle before the driver sends a KeepAlive packet. This property applies to Linux and MacOS only. On Microsoft Windows this value is determined by the operating system.

Property	Value	Default	Description
`tcpKeepAliveInterval`	`<seconds>` This value is ignored if it is 0.	4	Controls the number of specified seconds before the driver sends an additional KeepAlive packet if there was no response to the previous KeepAlive packet. This property applies to Linux and MacOS only. On Microsoft Windows this value is determined by the operating system.
`timeZonePerObject`	Boolean	TRUE	Instantiates a new java.util.TimeZone object for each prepared statement and result set, for use in time-zone conversions. The default is TRUE, which allows the driver to react to time zone changes (such as when switching to and from daylight savings time) while the JVM is running. Specifying FALSE can significantly reduce the number of object instantiations in the JVM, thus reducing the frequency with which the garbage collector must run.

Property	Value	Default	Description
`traceFile`	`<JDBC-trace-file-name>`	`traceFile` and `traceOptions` override the HDB_JDBC_TRACEFILE and HDB_JDBC_TRACEOPTIONS environment variables. Environment variables override standard trace settings file support.	Implicitly enables tracing for the current JDBC connection only when combined with `traceOptions` connection property. The special file names "stdout" and "stderr" are supported to send trace information to stdout and stderr, respectively. In these cases, all trace lines are prefixed with the tracer class name and hash code, in order to help identify the JDBC trace lines; for example, "[com.sap.db.jdbc.trace.Tracer@7ec02cfd]". The trace file name supports %a and %c replacement parameters to support multiple trace files. For more information regarding these parameters, see JDBC Tracing and Trace Options [page 491]
`traceOptions`	CONNECTIONS, API, PACKET, DISTRIBUTION, STATISTICS, CLEANERS, DEBUG for SAP HANA client versions 2.4 and lower CONNECTIONS, API, PACKET, DISTRIBUTION, STATISTICS, CLEANERS, DEBUG, PLAINTEXTCSE, TIMESTAMPS, ELAPSEDTIMES for SAP HANA client versions 2.5 and higher	`traceFile` and `traceOptions` override the `HDB_JDBC_TRACEFILE` and `HDB_JDBC_TRACEOPTIONS` environment variables. Environment variables override standard trace settings file support.	Implicitly enables tracing for the current JDBC connection only when combined with `traceFile` connection property. Lists values in a comma-separated, case-insensitive list. For example, "Connections,API,packet,timestamps" enables the CONNECTIONS, API, and PACKET trace levels (while leaving the remaining trace levels disabled), and also marks each trace entry with a timestamp.

Property	Value	Default	Description
`transactionalLobs`	Boolean	TRUE	Simulates LOB behavior as required by the JDBC specification by postponing the closing of statements and result sets that reference LOBs until the end of the current transaction.
`user`	`<user>`	Empty	Specifies the user name. This property is optional depending on the authentication method used.
`virtualHostName`	`<virtual-host-name>`	Empty	Specifies the virtual host name. This value is ignored if no hdbuserstore key is specified.
`webSocketURL` (SAP HANA Service only)	`<websocket-URL>`	Empty	Specifies the WebSocket URL used to connect to a single-node SAP HANA server. The connection property value cannot be empty. WebSockets connections have the following default connection property settings: • encrypt=TRUE • compress=TRUE • ignoreTopology=TRUE • reconnect=FALSE

Security Connection Properties

Property	Value	Default	Description
`authenticationMethods`	`<comma-separated-string>`		Provides a comma-separated list of authentication methods to be used by the client. Possible values are: password, pbkdf2, ldap, saml, jwt, saplogon, kerberos, sessioncookie, and X509.

Property	Value	Default	Description
`authenticationX509`	`<string>`	`<value-of-sslKeyStore>`	Provides a string containing the X509 certificate or the name of a file containing the X509 certificate. The certificate must be PEM-encoded in PKCS8 format. Defaults to the existing value of `sslKeyStore`, if specified.
`cseKeyStorePassword`	`<string>`	Empty	Provides the password for the local key store. This option is required when using client-side encryption. This must be at least eight characters and have at least one upper case character, one lower case character, and one number.
`encrypt`	Boolean	FALSE	Enables or disables TLS encryption. For SAP HANA Cloud connections to succeed, encrypt must be set to TRUE.

Property	Value	Default	Description
`hostNameInCertificate`	`<string>`	Empty	Specifies the host name used to verify the server's identity. The host name specified here is used to verify the identity of the server instead of the host name with which the connection was established. For example, in a single-host system, if a connection is established from a client on the same host as the server, a mismatch would arise between the host named in the certificate (actual host name) and the host used to establish the connection (localhost). **Note** If you specify * as the host name, this property has no effect. Other wildcards are not permitted.
`keyStore`	`<file>` | `<store name>`	`<VM default>`	Specifies the location of the Java keystore.
`keyStorePassword`	`<password>`	`<VM default>`	Specifies the password used to access the private key from the keystore file. **Note** This property is not used for SAP HANA studio connections.
`keyStoreType`	`<JKS>` | `<PKCS12>`	`<VM default>`	Specifies the Java keystore file format.
`sniHostname`	`<string>`	Empty	Specifies the name of the host that is attempting to connect at the start of the TLS handshaking process.

Property	Value	Default	Description
`sslMinProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	TLS12	Controls the minimum TLS version. The lowest protocol version is accepted in TLS handshakes.
`sslMaxProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	MAX	Controls the maximum TLS version. MAX allows this parameter to be future-proof in the event that new TLS versions are supported. The highest negotiated protocol version is used in TLS handshakes.
`sslKeyStore`	`<string>`	Empty	Specifies an alternative to the `keyStore` connection property. This property allows you to specify the contents of the keystore file as a string. The certificates and private keys must be PEM-encoded, and the private keys must be in PKCS8 format.
`sslSNIRequest`	Boolean	TRUE	Specifies whether SNI requests are enabled for TLS connections: TRUE/FALSE.
`sslTrustStore`	`<string>`	Empty	Specifies an alternative to the `trustStore` connection property. This property allows you to specify the contents of the trust store file as a string. The certificates must be PEM-encoded.

Property	Value	Default	Description
`trustStore`	`<file>|<store name>`	`<VM default>`	Specifies the path to the trust store file that contains the server's public certificate(s). Typically, the trust store contains the root certificate or the certificate of the certification authority that signed the server's certificate(s).
`trustStorePassword`	`<password>`	`<VM default>`	Specifies the password used to access the trust store file.
`trustStoreType`	`<JKS>`	`<VM default>`	Specifies the file format of the trust store file.
`validateCertificate`	Boolean	TRUE	If set to true, specifies that the server's certificate is validated.

Related Information

Setting Session-Specific Client Information [page 16]
JDBC Tracing and Trace Options [page 491]
Environment Variables [page 20]

4.3.1.3.1 How To Set JDBC Connection Properties

The following example sets JDBC connection properties that supply a user and password and disable all communication timeouts.

```
Properties p = new Properties();
p.setProperty( "user", user );
p.setProperty( "password", password );
p.setProperty( "latency", "0" );
p.setProperty( "communicationtimeout", "0" );
connection = DriverManager.getConnection( url, p );
```

Alternately, connection properties can be added to the URL directly, for example:

```
jdbc:sap://localhost:30015/?
user=myUser&password=myPassword&latency=0&communicationtimeout=0
```

4.3.1.4 Valid Java-to-SQL Conversions

SAP HANA allows each Java object to be converted to specific SQL types using the JDBC method `PreparedStatement.setObject` or `RowSet.setObject`.

Some conversions may fail at runtime if the value passed is invalid.

	Boolean	Byte	Short	Int	Long	Float	Double	BigDecimal	Date	Time	Timestamp	String	byte[]	Clob	NClob	Blob
BOOLEAN	X	X	X	X	X	X	X	X				X				
TINYINT	X	X	X	X	X	X	X	X				X				
SMALLINT	X	X	X	X	X	X	X	X				X				
INTEGER	X	X	X	X	X	X	X	X				X				
BIGINT	X	X	X	X	X	X	X	X				X				
REAL	X	X	X	X	X	X	X	X				X				
DOUBLE	X	X	X	X	X	X	X	X				X				
DECIMAL	X	X	X	X	X	X	X	X				X				
SMALLDECIMAL	X	X	X	X	X	X	X	X				X				
DATE / DAYDATE									X		X	X				
TIME / SECONDTIME										X	X	X				

SECONDDATE									X	X	X	X				
TIMESTAMP/LONGDATE									X	X	X	X				
CHAR	X	X	X	X	X	X	X	X	X	X	X	X	X			
VARCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X			
NCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X			
NVARCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X			
SHORTTEXT	X	X	X	X	X	X	X	X	X	X	X	X	X			
BINARY												X	X			
VARBINARY												X	X			
ST_POINT												X	X			
ST_GEOMETRY												X	X			
CLOB												X	X	X	X	

NCLOB	X	X	X	X	
BLOB	X	X			X
TEXT	X	X	X	X	
BINTEXT	X	X	X	X	
ARRAY					

4.3.2 Cursor Holdability Support

Cursor holdability specifies whether the cursor for a result set remains open after the transaction is committed or rolled back.

If a cursor remains open, then the application is able to fetch more rows from it.

Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources.

The JDBC standard implements two values for cursor behavior: HOLD_CURSORS_OVER_COMMIT and CLOSE CURSORS_AT_COMMIT.

The SAP HANA JDBC driver supports additional values to configure cursor behavior: HOLD_CURSORS_OVER_ROLLBACK and HOLD_CURSORS_OVER_COMMIT_AND_ROLLBACK. Support for holding a cursor over rollback or over commit and rollback is available in SAP HANA servers beginning with SAP HANA 2.0 SPS 04.

The following methods support the HOLD_CURSORS_OVER_ROLLBACK and HOLD_CURSORS_OVER_COMMIT_AND_ROLLBACK cursor holdability values:

- Connection.getHoldability()
- Connection.setHoldability()
- Connection.createStatement()
- Connection.prepareStatement()
- Connection.prepareCall()
- DatabaseMetaData.supportsResultSetHoldability()
- DatabaseMetaData.getResultSetHoldability()
- Statement.getResultSetHoldability()
- ResultSet.getHoldability()

4.3.3 JDBC Data Source Example for Use with Apache Tomcat

This sample illustrates how to use the SAP HANA JDBC driver's data source classes in a web application, to be deployed to an Apache Tomcat server.

The SAP HANA JDBC driver provides three data source classes:

1. `com.sap.db.jdbcext.HanaDataSource`: An implementation of the javax.sql.DataSource interface
2. `com.sap.db.jdbcext.HanaConnectionPoolDataSource`: An implementation of the javax.sql.ConnectionPoolDataSource interface
3. `com.sap.db.jdbcext.HanaXADataSource`: An implementation of the javax.sql.XADataSource interface

This web application defines a data source instance for each data source type and establishes connections to each data source.

1. Source Files

The web application consists of the following four files:

web.xml

Defines the data source names, descriptions, and class names.

Example:

```
<web-app
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://
java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
<resource-ref>
    <description>HANA DataSource</description>
    <res-ref-name>jdbc/basicConnectTest</res-ref-name>
    <res-type>com.sap.db.jdbcext.HanaDataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
<resource-ref>
    <description>HANA ConnectionPoolDataSource</description>
    <res-ref-name>jdbc/pooledConnectTest</res-ref-name>
    <res-type>com.sap.db.jdbcext.HanaConnectionPoolDataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
<resource-ref>
    <description>HANA XADataSource</description>
    <res-ref-name>jdbc/xaConnectTest</res-ref-name>
    <res-type>com.sap.db.jdbcext.HanaXADataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
</web-app>
```

context.xml

- Defines the connection properties for the data sources in web.xml
- Replace the values for hostlist, user, and password with the values for your database

Example:

```
<Context>
    <Resource
        name="jdbc/basicConnectTest"
        type="com.sap.db.jdbcext.HanaDataSource"
        factory="com.sap.db.jdbcext.DataSourceSAPFactory"
        driverClassName="com.sap.db.jdbc.Driver"
        hostlist="myserver.sap:30015"
        user="system"
        password="manager"/>
    <Resource
        name="jdbc/pooledConnectTest"
        type="com.sap.db.jdbcext.HanaConnectionPoolDataSource"
        factory="com.sap.db.jdbcext.DataSourceSAPFactory"
        driverClassName="com.sap.db.jdbc.Driver"
        hostlist="myserver.sap:30015"
        user="system"
        password="manager"/>
    <Resource
        name="jdbc/xaConnectTest"
        type="com.sap.db.jdbcext.HanaXADataSource"
        factory="com.sap.db.jdbcext.DataSourceSAPFactory"
        driverClassName="com.sap.db.jdbc.Driver"
        hostlist="myserver.sap:30015"
        user="system"
        password="manager"/>
</Context>
```

index.jsp

Instantiates the application class and invokes its methods using the data sources.

Example:

```
<html>
<head><title>HANA ConnectTest</title></head>

<body>
<p>HANA ConnectTest</p>
<%@ page import="javax.naming.*" %>
<%@ page import="mytest.*" %>

<%
Context       context       = new InitialContext();
ConnectTest connectTest      = new ConnectTest( out, context );

connectTest.testDataSource( "java:/comp/env/jdbc/basicConnectTest",
ConnectTest.DataSourceType.BASIC );
connectTest.testDataSource( "java:/comp/env/jdbc/pooledConnectTest",
ConnectTest.DataSourceType.POOLED );
connectTest.testDataSource( "java:/comp/env/jdbc/xaConnectTest",
ConnectTest.DataSourceType.XA );
%>
</body>
</html>
```

ConnectTest.java

The JDBC application source code.

Example:

```
//
************************************************************************
// Copyright (c) 2023 SAP SE or an SAP affiliate company. All rights
reserved.
```

```
//
************************************************************************

package mytest;

import java.io.*;
import java.sql.*;
import javax.naming.*;
import javax.sql.*;

public class ConnectTest
{
       // Data source types
       public enum DataSourceType
       {
              BASIC, POOLED, XA
       }

       private static final int    NUM_ITERATIONS = 3;
       private static final String SQL_SELECT     = "SELECT
LOGICAL_CONNECTION_ID FROM M_CONNECTIONS WHERE OWN = 'TRUE'";

       private final Writer  _writer;
       private final Context _context;

       public ConnectTest( Writer writer, Context context )
       {
              _writer = writer;
              _context = context;
       }

       public void testDataSource( String dataSourceName, DataSourceType
dataSourceType ) throws Exception
       {
              DataSource               dataSource               = null;
              ConnectionPoolDataSource connectionPoolDataSource = null;
              PooledConnection         pooledConnection         = null;
              XADataSource             xaDataSource             = null;
              XAConnection             xaConnection             = null;
              Connection               conn;
              PreparedStatement        ps;
              ResultSet                rs;
              int                      connId;

              // Get the data source and if applicable, the pooled or XA
connection
              switch( dataSourceType ) {
              case BASIC:
                     try {
                           dataSource = (DataSource)
_context.lookup( dataSourceName );
                     } catch( NamingException e ) {
                           _fail( e, "Can't get data source with name \""
+ dataSourceName + '"' );
                     }
                     break;

              case POOLED:
                     try {
                           connectionPoolDataSource =
(ConnectionPoolDataSource) _context.lookup( dataSourceName );
                     } catch( NamingException e ) {
                           _fail( e, "Can't get connection pool data
source with name \"" + dataSourceName + '"' );
                     }
                     try {
```

```
                        pooledConnection =
connectionPoolDataSource.getPooledConnection();
                } catch( SQLException e ) {
                        _fail( e, "Can't get pooled connection" );
                }
                break;

            case XA:
                try {
                        xaDataSource = (XADataSource)
_context.lookup( dataSourceName );
                } catch( NamingException e ) {
                        _fail( e, "Can't get XA data source with name
\"" + dataSourceName + '"' );
                }
                try {
                        xaConnection = xaDataSource.getXAConnection();
                } catch( SQLException e ) {
                        _fail( e, "Can't get XA connection" );
                }
                break;

            default:
                throw new AssertionError( "Unexpected data source
type: " + dataSourceType );
            }

            for( int iteration = 1; iteration <= NUM_ITERATIONS;
iteration++ ) {

                conn = null;
                ps = null;
                rs = null;
                try {
                        // Get the connection
                        try {
                                switch( dataSourceType ) {
                                case BASIC:
                                        conn = dataSource.getConnection();
                                        break;
                                case POOLED:
                                        conn =
pooledConnection.getConnection();
                                        break;
                                case XA:
                                        conn =
xaConnection.getConnection();
                                        break;
                                default:
                                        throw new
AssertionError( "Unexpected test type: " + dataSourceType );
                                }
                        } catch( SQLException e ) {
                                _fail( e, "Can't get connection" );
                        }

                        // Prepare and execute the query
                        ps = conn.prepareStatement( SQL_SELECT );
                        rs = ps.executeQuery();
                        while( rs.next() ) {
                                connId = rs.getInt( 1 );
                                _message( dataSourceName + " " +
iteration + " " + connId );
                        }
                } finally {
                        // Cleanup
                        _close( rs );
```

```
                        _close( ps );
                        _close( conn );
                    }
                }

                // Close the pooled or XA connection
                switch( dataSourceType ) {
                case BASIC:
                    break;
                case POOLED:
                    pooledConnection.close();
                    break;
                case XA:
                    xaConnection.close();
                    break;
                default:
                    throw new AssertionError( "Unexpected test type: " +
dataSourceType );
                }
        }

        private void _close( AutoCloseable autoCloseable ) throws Exception
        {
                autoCloseable.close();
        }

        private void _message( String message )
        {
                System.err.println( message );

                if( _writer != null ) {
                    try {
                        _writer.write( "<p>" + message + "</p>" );
                    } catch( IOException e ) {
                        // ignore
                    }
                }
        }

        private void _fail( Exception e, String message ) throws Exception
        {
                _message( message );
                throw e;
        }
}
```

2. Deploying the Application

These instructions assume you have installed Apache Tomcat in a directory defined by the $CATALINA_HOME environment variable.

1. Create a subdirectory for your application in the $CATALINA_HOME/webapps directory; for example `$CATALINA_HOME/webapps/connectTest`
2. Create META-INF and WEB-INF subdirectories in this application subdirectory
3. Copy index.jsp to connectTest, context.xml to connectTest/META-INF, and web.xml to connectTest/WEB-INF
4. Compile ConnectTest.java, create a JAR file containing the compiled classes, and copy the JAR to your $CATALINA_HOME/lib directory

3. Running the Application

1. Restart your Tomcat server. Here we assume the server is running on localhost:8080
2. Execute the web application by opening the URL http://localhost:8080/connecttest/. The expected output is:

```
HANA ConnectTest
java:/comp/env/jdbc/basicConnectTest 1 316628
java:/comp/env/jdbc/basicConnectTest 2 316629
java:/comp/env/jdbc/basicConnectTest 3 316630
java:/comp/env/jdbc/pooledConnectTest 1 316631
java:/comp/env/jdbc/pooledConnectTest 2 316631
java:/comp/env/jdbc/pooledConnectTest 3 316631
java:/comp/env/jdbc/xaConnectTest 1 316632
java:/comp/env/jdbc/xaConnectTest 2 316632
java:/comp/env/jdbc/xaConnectTest 3 316632
```

The application establishes three connections for each data source and displays the connection ID of each connection.

> **Note**
>
> The basic data source establishes a new connection each time, while the connection pool and XA data sources reuse the previous connection.

4.3.4 JDBC Driver Static Methods

The SAP HANA JDBC driver (com.sap.db.jdbc.Driver) supports static methods.

Return Type	Method	Description
int	getAnchorConnectionID(java.sql.Connection) Method [page 477]	Returns the anchor connection ID.
int	getPacketSize(java.sql.Statement) Method [page 478]	Returns the packet size for a statement.
int	getPrimaryConnectionID(java.sql.Connection) Method [page 479]	Returns the primary connection ID.
List<java.lang.String>	getPrintLines(java.sql.Statement) Method [page 479]	Allows applications to access to messages printed from a stored procedure via SQLSCRIPT_PRINT:PRINT_LINE.
long	getServerCPUTime(java.sql.ResultSet) Method [page 481]	Retrieves the total server CPU processing time.
long	getServerCPUTime(java.sql.Statement) Method [page 481]	Indicates whether the specified date value is the out-of-band value that corresponds to an empty string.
long	getServerMemoryUsage(java.sql.ResultSet) Method [page 482]	Returns memory usage data for a result set.

Return Type	Method	Description
long	getServerMemoryUsage(java.sql.Statement) Method [page 483]	Returns memory usage data for a prepared or executed statement.
long	getServerProcessingTime(java.sql.ResultSet) Method [page 484]	Returns the elapsed processing time for a result set.
long	getServerProcessing-Time(java.sql.Statement) Method [page 485]	Returns the elapsed processing time for a prepared or executed statement.
boolean	isEmptyDate(java.sql.Date) Method [page 486]	Indicates whether the specified date value is the out-of-band value that corresponds to an empty string.
boolean	isEmptyTime(java.sql.Time) Method [page 487]	Indicates whether the specified time value is the out-of-band value that corresponds to an empty string.
boolean	isEmptyTimestamp(java.sql.Timestamp) Method [page 487]	Indicates whether the specified timestamp value is the out-of-band value that corresponds to an empty string.
void	setPacketSize(java.sql.Statement, int) Method [page 488]	Sets the packet size for a statement.
void	suspendTracing(java.sql.Connection) Method [page 489]	Suspends JDBC tracing on a per connection basis. Specifying NULL as the connection argument suspends tracing for all connections in the JVM.
void	resumeTracing(java.sql.Connection) Method [page 489]	Resumes JDBC tracing on a per connection basis. Specifying NULL as the connection argument resumes tracing for all connections in the JVM.

Related Information

JDBC Tracing and Trace Options [page 491]

4.3.4.1 getAnchorConnectionID(java.sql.Connection) Method

Returns the ID of the initial statement routing connection that was made when the application established the connection.

Syntax

```
public static int getAnchorConnectionID( java.sql.Connection <connection> )
```

Parameters

Type	Name	Description
`java.sql.Connection`	`connection`	The connection whose ID you are requesting.

Returns

Type	Description
int	The ID of the initial statement routing connection.

4.3.4.2 getPacketSize(java.sql.Statement) Method

Returns the packet size for a statement.

Syntax

```
public static int getPacketSize( java.sql.Statement <statement> )
```

Parameters

Type	Name	Description
`java.sql.Statement`	`statement`	The statement whose packet size you are requesting.

Returns

Type	Description
int	If the packet size was not set via `setPacketSize(java.sql.Statement)`, this function returns the default packet size or the value of the connection's `packetSize` connect property (if specified).

4.3.4.3 getPrimaryConnectionID(java.sql.Connection) Method

Returns the primary connection ID.

Syntax

```
public static int getPrimaryConnectionID( java.sql.Connection <connection> )
```

Parameters

Type	Name	Description
`java.sql.Connection`	`connection`	The connection whose primary connection ID you are requesting.

Returns

Type	Description
int	The ID of the primary connection.

4.3.4.4 getPrintLines(java.sql.Statement) Method

Allows applications to access to messages printed from a stored procedure via SQLSCRIPT_PRINT:PRINT_LINE.

Syntax

```
public static java.util.List<java.lang.String> getPrintLines(
java.sql.Statement <statement> )
```

Parameters

Type	Name	Description
`java.sql.Statement`	`statement`	The java.sql.Statement, java.sql.PreparedStatement, or java.sql.CallableStatement that executed the procedure.

Returns

Type	Description
List<java.lang.String>	Each string in the unmodifiable list is a message printed from the procedure. The strings appear in the list in the order that they were printed. If no messages were printed, then the method returns an empty list of strings.

Remarks

This static method on com.sap.db.jdbc.driver makes messages printed from a stored procedure via SQLSCRIPT_PRINT:PRINT_LINE accessible to applications.

Example

The following example returns strings from the PrintAPITest1 procedure:

```
import java.sql.*;
import java.util.*;
import com.sap.db.jdbc.driver;
...
Connection      conn = ...
Statement       stmt = conn.createStatement();
List<String>    printLines;
stmt.executeUpdate( "CREATE PROCEDURE PrintAPITest1 AS\n"
     + "BEGIN\n"
     + "    USING SQLSCRIPT_PRINT AS PRINT;\n"
     + "    PRINT:PRINT_LINE('Hello, world');\n"
     + "    PRINT:PRINT_LINE('Line 2');\n"
     + "END;" );
stmt.executeUpdate( "CALL PrintAPITest1()" );
printLines = driver.getPrintLines( stmt );
// printlines contains two Strings: "Hello, world" and "Line 2"
```

4.3.4.5 getServerCPUTime(java.sql.ResultSet) Method

Returns the server CPU time for the result set fetch.s

Syntax

```
public static long getServerCPUTime( java.sql.ResultSet <resultSet> )
```

Parameters

Type	Name	Description
`java.sql.ResultSet`	`resultSet`	The result set whose CPU fetch time you are requesting.

Returns

Type	Description
long	The server CPU time for the result set fetch, in microseconds. This value does not include network communication time.

Remarks

The returned values are for the fetches since the beginning of the current result set. If nextResult is called, then the returned results are reset for the new result set. By default, executed statements include fetch times and memory usage data for the first 32 rows.

4.3.4.6 getServerCPUTime(java.sql.Statement) Method

Retrieves the total server CPU processing time.

Syntax

```
public static long getServerCPUTime( java.sql.Statement <statement> )
```

Parameters

Type	Name	Description
`java.sql.Statement`	`statement`	The statement whose CPU fetch time you are requesting.

Returns

Type	Description
long	The server CPU processing time that it took to prepare or execute statements, in microseconds. This value does not include network communication time.

Remarks

If a statement has been prepared but not executed, then the returned value is only for the prepare. If the statement has been executed, then the returned value is only for the most recent execution. By default, the execution server CPU time includes the fetch server CPU time for the first 32 rows.

4.3.4.7 getServerMemoryUsage(java.sql.ResultSet) Method

Returns the server memory usage for the result set fetches.

Syntax

```
public static long getServerMemoryUsage( java.sql.ResultSet <resultSet> )
```

Parameters

Type	Name	Description
`java.sql.ResultSet`	`resultSet`	The result set whose memory usage you are requesting.

Returns

Type	Description
long	The server memory usage for the result set fetches, in bytes.

Remarks

The returned value is for the fetches since the beginning of the current result set. If nextResult is called, then the returned results are reset for the new result set. By default, executed statements include server memory usage for the first 32 rows. If fetching more rows requires additional fetch requests to the server, then the server memory usage value is a non-zero value that is calculated from the beginning of the current result set.

> **Note**
>
> `getServerMemoryUsage` returns 0 if you are connected to a HANA 2 SPS 03 or earlier server.
>
> `getServerMemoryUsage` returns 0 unless memory_tracking and resource_tracking is enabled in the resource_tracking section of `global.ini` by setting enable_tracking and memory_tracking to on. For performance reasons, resource tracking is disabled by default.

4.3.4.8 getServerMemoryUsage(java.sql.Statement) Method

Returns server memory usage for a prepared or executed statement.

> **Syntax**

```
public static long getServerMemoryUsage( java.sql.Statement <statement> )
```

Parameters

Type	Name	Description
`java.sql.Statement`	`statement`	The statement whose memory usage you are requesting.

Returns

Type	Description
long	Memory usage for prepared or executed statements on the server side, in bytes.

Remarks

If a statement has been prepared but not executed, then the returned value is only for the prepare. If the statement has been executed, then the returned value is only for the most recent execution. By default, executed statements include fetchserver memory usage for the first 32 rows.

> Note
>
> `getServerMemoryUsage` returns 0 if you are connected to a HANA 2.0 SPS 03 or earlier server.
>
> `getServerMemoryUsage` returns 0 unless memory_tracking and resource_tracking is enabled in the resource_tracking section of `global.ini` by setting enable_tracking and memory_tracking to on. For performance reasons, resource tracking is disabled by default.

4.3.4.9 getServerProcessingTime(java.sql.ResultSet) Method

Returns the elapsed server processing time for the result set fetches.

> Syntax

```
public static long getServerProcessingTime( java.sql.ResultSet <resultSet> )
```

Parameters

Type	Name	Description
`java.sql.ResultSet`	`resultSet`	The result set whose server processing time you are requesting.

Returns

Type	Description
long	The elapsed processing time for a result set on the server side, in microseconds. This value does not include network communication time.

Remarks

The returned value is for the fetches since the beginning of the current result set. If nextResult is called, then the returned result is reset for the new result set. By default, executed statements include fetch server processing time for the first 32 rows. If fetching more rows requires additional fetch requests to the server, then the server processing time is a non-zero value and calculated from the beginning of the current result set.The elapsed server processing time for the result set fetches, in microseconds. This does not include network communication time.

4.3.4.10 getServerProcessingTime(java.sql.Statement) Method

Returns the elapsed server processing time for a prepared or executed statement.

Syntax

```
public static long getServerProcessingTime( java.sql.Statement <statement> )
```

Parameters

Type	Name	Description
`java.sql.Statement`	`statement`	The statement server processing time you are requesting.

Returns

Type	Description
long	The elapsed processing time for a prepared or executed statement on the server side, in microseconds. This value does not include network communication time.

Remarks

If a statement has been prepared but not executed, then the returned value is only for the prepared statement. If the statement has been executed, then the returned value is only for the most recent execution.

By default, executed statements include the server fetch processing time for the first 32 rows.

4.3.4.11 isEmptyDate(java.sql.Date) Method

Indicates whether the specified date value is the out-of-band value that corresponds to an empty String.

Syntax

```
public static boolean isEmptyDate( java.sql.Date <date> )
```

Parameters

Type	Name	Description
`java.sql.Date`	`date`	The date that you are checking for whether it is an out-of-band value.

Returns

Type	Description
boolean	Returns TRUE if the specified java.sql.Date is an out-of-band value; otherwise, it returns FALSE.

Related Information

JDBC Connection Properties [page 441]

4.3.4.12 isEmptyTime(java.sql.Time) Method

Indicates whether the specified time value is the out-of-band value that corresponds to an empty string.

Syntax

```
public static boolean isEmptyTime( java.sql.Time <time> )
```

Parameters

Type	Name	Description
`java.sql.Time`	`time`	The time that you are checking for whether it is an out-of-band value.

Returns

Type	Description
boolean	Returns TRUE if the specified java.sql.Time is an out-of-band value; otherwise, it returns FALSE.

Related Information

JDBC Connection Properties [page 441]

4.3.4.13 isEmptyTimestamp(java.sql.Timestamp) Method

Indicates whether the specified timestamp value is the out-of-band value that corresponds to an empty string.

Syntax

```
public static boolean isEmptyTimestamp( java.sql.Timestamp <timestamp> )
```

Parameters

Type	Name	Description
`java.sql.Timestamp`	`timestamp`	The timestamp that you are checking for whether it is an out-of-band value.

Returns

Type	Description
boolean	Returns TRUE if the specified java.sql.Timestamp is an out-of-band value; otherwise, it returns FALSE.

Related Information

JDBC Connection Properties [page 441]

4.3.4.14 setPacketSize(java.sql.Statement, int) Method

Sets the packet size for a statement.

Syntax

```
public static void setPacketSize( java.sql.Statement <statement>, <int> )
```

Parameters

Type	Name	Description
`java.sql.Statement`	`statement`	The statement whose packet size you are setting.
value	`int`	The value of the packet size.

Remarks

The value must be greater than the default packet size or the connection's `packetSize` connection property, otherwise the default packet size or specified connection packet size is used.

Specifying a per-statement packet size improves query performance by reducing the number of packets required to complete the following operations:

- Batch inserts, updates, and deletes: controls the Request Packet Size used
- WriteLOB requests: controls the Request Packet Size used
- ReadLOB requests: controls the Read LOB chunk size of data requested (in bytes)
- ResultSet fetches: controls the number of rows being fetched (packet size/estimation of the row size)

4.3.4.15 suspendTracing(java.sql.Connection) Method

Suspends JDBC tracing on a per connection basis. Specifying NULL as the connection argument suspends tracing for all connections in the JVM.

Syntax

```
public static void suspendTracing( java.sql.Connection <connection> )
```

Parameters

Type	Name	Description
`java.sql.Connection`	`connection`	Specifies the connection ID being requested.

4.3.4.16 resumeTracing(java.sql.Connection) Method

Resumes JDBC tracing on a per connection basis. Specifying NULL as the connection argument resumes tracing for all connections in the JVM.

Syntax

```
public static void resumeTracing( java.sql.Connection <connection> )
```

Parameters

Type	Name	Description
`java.sql.Connection`	`connection`	Specifies the connection ID being requested.

4.3.5 Savepoint Support

You can define savepoints in a transaction to separate groups of related statements.

The SAP HANA JDBC driver supports the following APIs defined by the JDBC specification to implement savepoint support:

Class	Method
java.sql.Connection	`Savepoint setSavepoint() throws SQLException`
	`Savepoint setSavepoint( String <name> ) throws SQLException`
	`void rollback( Savepoint <savepoint> ) throws SQLException`
	`void releaseSavepoint( Savepoint <savepoint> ) throws SQLException`
java.sql.DatabaseMetaData	`boolean supportsSavepoints() throws SQLException`
java.sql.Savepoint	`int getSavepointId() throws SQLException`
	`String getSavepointName() throws SQLException`

The APIs are implemented by using the following SQL statements:

- SAVEPOINT `<savepoint_name>`
- RELEASE SAVEPOINT `<savepoint_name>`
- ROLLBACK TO SAVEPOINT `<savepoint_name>`

Related Information

SAVEPOINT Statement (Transaction Management)
RELEASE SAVEPOINT Statement (Transaction Management)
ROLLBACK TO SAVEPOINT Statement (Transaction Management)

4.3.6 JDBC Tracing and Trace Options

Activate JDBC tracing to find errors while your application is connected to a database via JDBC. To enable JDBC tracing, you must use trace options.

The first column shows the field name in the SAP HANA JDBC Driver Trace Configuration tool, and the second column shows the command to enter when using the command-line tool.

Trace Options

Option	Command Line Option	Description
Enable tracing	`TRACE ON | OFF`	Starts and stops tracing.
Trace file name	`TRACE FILENAME [<path>]<file_name>`	Sets the name of the trace file. The system assigns each trace file an additional unique ID, so the file name is: `<file_name><ID>.<extension>`. The default file name is `jdbctrace` and the default extension is `prt`. The special file names "stdout" and "stderr" are supported to send trace information to stdout and stderr, respectively. The additional unique ID is not appended to the file name. In these cases, all trace lines are prefixed with the tracer class name and hash code, in order to help identify the JDBC trace lines; for example, "[com.sap.db.jdbc.trace.Tracer@7ec02cfd]". The trace file name supports %a and %c replacement parameters to support multiple trace files. For more information regarding these parameters, see the "Remarks" section.
Connections	`TRACE CONNECTIONS ON | OFF`	Enables/disables tracing for connections and disconnections. This option has no effect until tracing is enabled by using the *Enable tracing* field or the `TRACE ON` command.
JDBC APIs	`TRACE API ON | OFF`	Enables/disables JDBC API tracing. This option has no effect until tracing is enabled by using the *Enable tracing* field or the `TRACE ON` command.

Option	Command Line Option	Description
Packets	`TRACE PACKET ON | OFF`	Enables/disables communication packet tracing. This option has no effect until tracing is enabled by using the *Enable tracing* field or the `TRACE ON` command.
Distribution	`TRACE DISTRIBUTION ON | OFF`	Enables/disables distribution feature tracing. This option has no effect until tracing is enabled by using the *Enable tracing* field or the `TRACE ON` command.
Statistics	`TRACE STATISTICS ON | OFF`	Enables/disables trace of connection statistics, including durations, packet and byte counts, and compression ratios. This option has no effect until tracing is enabled by using the *Enable tracing* field or the `TRACE ON` command.
Cleaners	`TRACE CLEANERS ON | OFF`	Enables/disables cleaners/finalizers for connections, statements, and result set tracing. This option has no effect until tracing is enabled by using the *Enable tracing* field or the `TRACE ON` command.
Debug	`TRACE DEBUG ON | OFF`	Enables/disables JDBC driver debug message tracing. This option has no effect until tracing is enabled by using the *Enable tracing* field or the `TRACE ON` command.
Encrypted values	`TRACE SHOW PLAINTEXTCSE ON | OFF`	Enables/disables client-side encrypted value tracing, in plain text. By default, client-side encrypted values are shown as "***" in the trace files. To show the actual unencrypted values, set this option to ON. This option has no effect until tracing is enabled by using the *Enable tracing* field or the `TRACE ON` command.
Show timestamps	`TRACE SHOW TIMESTAMPS ON | OFF`	Enables/disables timestamps for each trace record.

Option	Command Line Option	Description
Show elapsed times	`TRACE SHOW ELAPSEDTIMES ON | OFF`	Enables/disables elapsed times for each JDBC API call trace record and each packet send/receive record.

Limit the trace file size	`TRACE SIZE <size> [KB | MB | GB]`
	To remove the size limit, use the following option:
	`TRACE SIZE UNLIMITED`
		Limits the size of each trace file. The minimum size is 8192 bytes.

Stop tracing on an error code	`TRACE STOP ON ERROR <error_code>`
	To no longer stop on the specified error, use the following option:
	`TRACE STOP ON ERROR OFF`
		Stops writing the JDBC trace when the specified error code occurs.

Report trace failures	`TRACE FAILURE ACTION IGNORE | STDOUT | STDERR | EXCEPTION`	Reports trace failures using the following command line options: • IGNORE: Ignores tracing failures. • STDOUT: Reports tracing failures to stdout. • STDERR: Reports tracing failures to stderr. • EXCEPTION: Gives a java.lang.RuntimeException when a trace failure occurs.
Performance trace	`PERFORMANCETRACE ON | OFF`	Enables/disables performance tracing.
Performance trace file name	`PERFORMANCETRACE FILENAME`	Sets the name of the performance trace file. The JDBC driver adds a unique suffix to the trace file name.
Display the trace settings	`SHOW [ALL | TRACESETTINGS]`	Displays the current trace settings.
Help	`java -jar <installation_path>\ngdbc.jar -h | --help | HELP`	Displays the command line help.
Version	`java -jar <installation_path>\ngdbc.jar -v | --version | VERSION`	Displays the version information.

Option	Command Line Option	Description
GUI tool	`java -jar <installation_path>\ngdbc.jar -g | --gui | GUI`	Runs a GUI tool to configure trace settings.

Remarks

The trace file name supports %a and %c replacement parameters.

Using the %a parameter allows you to trace all connections for a given "application," as defined by the APPLICATIONUSER session variable, to a single trace file. The %a parameter is replaced with the value of the APPLICATIONUSER session variable when tracing begins. The value can be specified before a connection is established via the `applicationUser` connection property.

Example

If the trace file name is defined as "jdbctrace_%a_.prt" then all connections that specify the connection property applicationUser=MYAPP are traced to the file "jdbctrace_MYAPP_###.prt", where ### is a series of random decimal digits.

Using the %c parameter allows you to trace each "connection," defined as an anchor connection along with all its statement-routed and hint-routed connections, to its own trace file. The %c parameter is replaced with the value of the java.sql.Connection object's hash code, followed by an underscore, followed by the connection ID for the anchor connection.

Note

The anchor connection ID isn't defined until the connection is established. As such, the value "000000" is used as a placeholder until the first few packets have been exchanged, at which point tracing switches to a new file that includes the anchor connection ID in the file name.

Example

If the trace file name is defined as "jdbctrace_%c_.prt" then a java.sql.Connection object with hash code "1a2b3c4d" will initially be traced to the file "jdbctrace_1a2b3c4d_000000_###.prt". Once the anchor connection ID is returned, say as the value 301125, tracing will be switched to the file "jdbctrace_1a2b3c4d_301125_###.prt". The trace file will also be switched anytime a reconnect occurs, as this establishes a new anchor connection with a new connection ID.

If both the %a and %c parameters are specified in the trace file name, then each connection is traced to its own file, where the file name includes the APPLICATIONUSER session variable value, java.sql.Connection hash code, and anchor connection ID.

Related Information

Trace a JDBC Connection Using a Connection String [page 495]
Trace a JDBC Connection Using the Command Line [page 496]
Trace a JDBC Connection Using the GUI [page 497]

4.3.6.1 Trace a JDBC Connection Using a Connection String

Start tracing by adding an option in the connection string when creating a JDBC connection.

Prerequisites

- Enabling the JDBC trace significantly reduces JDBC performance.
- Enabling tracing can result in sensitive information (such as connection properties, SQL text, and result set values) being included in the trace file.
- Enabling the `traceFile` and `traceOptions` connection properties allows tracing for a given connection without affecting other connections.

Context

When JDBC tracing is activated, the JDBC driver logs the following information on the client:

- JDBC API calls called by the JDBC application
- JDBC API call parameters
- JDBC driver static methods
- Executed SQL statements and their results

The location of the trace file is determined by the `traceFile` connection property. The trace options are determined by the `traceOptions` connection property. Trace files include the trace settings in the trace file header, for example:

```
Trace configuration:
Levels: API,CONNECTIONS,PACKET
Show plain-text client-side encrypted values: Disabled
Show timestamps: Enabled
Trace file size: 12 MB
Stop on error: Disabled
```

When tracing completes, the trace file header includes the settings that were used to perform the trace.

Procedure

1. Stop the application that you are going to trace. Add the `traceFile` and `traceOptions` options to the connection when creating a JDBC connection to the application.
2. Stop tracing.

Example

The following example connection string starts tracing:

```
jdbc:sap://localhost:30015/?
autocommit=false&traceFile=traceFile.txt&traceOptions=CONNECTIONS,API,PACKET,TIME
STAMPS,ELAPSEDTIMES,FAILURE_ACTION_STDERR
```

Next Steps

Review the contents of the trace files.

4.3.6.2 Trace a JDBC Connection Using the Command Line

Start tracing by running the tracing configuration tool and sending commands via the command line.

Context

Tracing via the command line enables you to start and configure tracing without stopping and restarting your application that is connected via JDBC.

Procedure

1. Display the current configuration by running the command:

   ```
   java -jar <installation-path>\ngdbc.jar SHOW
   ```

   On Linux or UNIX, use / instead of \.
2. Optionally, show the available trace command-line options:

   ```
   java -jar <installation-path>\ngdbc.jar -h
   ```

3. Select trace options by running the command:

```
java -jar <installation-path>\ngdbc.jar <option>
```

4. Start tracing by running the command:

```
java -jar <installation-path>\ngdbc.jar TRACE ON
```

4.3.6.3 Trace a JDBC Connection Using the GUI

Configure and start JDBC tracing by using the SAP HANA JDBC Driver Trace Configuration graphical user interface (GUI).

Context

Tracing via the GUI enables you to start and configure tracing without stopping and restarting your application that is connected via JDBC.

Procedure

1. Open the SAP HANA JDBC Driver Trace Configuration GUI by running the following command:

```
java -jar <installation_path>\ngdbc.jar
```

   On Linux or UNIX, use / rather than \.
2. Click *Enable tracing*.
3. Select and configurate your tracing options.
4. Click *OK*.

4.3.7 Hibernate Dialect for SAP HANA

Hibernate is an Object/Relational Mapping (ORM) framework concerned with data persistence as it applies to relational databases (via JDBC).

The Hibernate Dialect for SAP HANA is a contribution to the Hibernate open-source project provided by and actively maintained by SAP SE, and is not part of a licensed SAP HANA edition or option. It is provided under the terms of the Hibernate license.

The Hibernate Dialect for SAP HANA is available with Hibernate version 5.2.12 and later, and it is recommended that you use it with the SAP HANA JDBC driver version 2.2.1 or later.

Related Information

http://hibernate.org/community/license/
https://docs.jboss.org/hibernate/orm/5.2/javadocs/index.html?org/hibernate/dialect/AbstractHANADialect.html
https://developers.sap.com/group.hana-hibernate-getting-started.html
https://developers.sap.com/tutorials/hana-hibernate-simple.html

4.4 Node.js Application Programming

Use the Node.js API to connect to SAP HANA databases, issue SQL queries, and obtain result sets.

For a list of supported Node.js versions, see 3165810

The Node.js driver does not support being called from JavaScript worker threads. Attempting to do so may lead to incorrect behavior.

Working with Node.js

The examples in this section show how to perform common tasks with the Node.js driver.

> **Note**
>
> The connection properties `host` and `port` are interchangeable with the `serverNode` connection property.

Getting Started

A database connection object is created by calling createConnection. The connection to the on-premise SAP HANA database server is established by calling the connection object's connection method and passing an object representing connection parameters, for example:

```
var hana = require('@sap/hana-client');
var conn = hana.createConnection();
var conn_params = {
  serverNode  : 'myserver:30015',
  uID         : 'system',
  pwd         : 'manager'
};
conn.connect(conn_params, function(err) {
  if (err) throw err;
  conn.exec('SELECT Name, Description FROM Products WHERE ID = ?',
[301],
            function (err, result) {
    if (err) throw err;
    console.log('Name: ', result[0].Name, ', Description: ',
                result[0].Description);
    // output --> Name: Tee Shirt, Description: V-neck
    conn.disconnect();
  })
```

```
});
```

Connecting

A database connection object is created by specifying createConnection. The connection is established by calling the connection object's connect method and passing in an object representing connection parameters.

The following example shows a TCP/IP connection to the on-premise SAP HANA database server:

```
conn.connect({
  host: 'myserver',
  port: '30015',
  uid: 'system',
  pwd: 'manager'
});
```

Disconnecting

Disconnect from the database:

```
conn.disconnect(function(err) {
  if (err) throw err;
  console.log('Disconnected');
});
```

Direct Statement Execution

Direct statement execution is the simplest way to execute SQL statements. The input consists of the SQL command to be executed and an optional array of positional arguments. The result is returned by using callbacks. The type of returned result depends on the kind of statement.

DDL Statement

In the case of a successful DDL statement, nothing is returned.

```
conn.exec('CREATE TABLE Test (ID INTEGER PRIMARY KEY, msg
VARCHAR(128))',
  function (err, result) {
  if (err) throw err;
  console.log('Table Test created!');
});
```

DML Statement

In the case of a DML statement the number of 'affectedRows' is returned.

```
conn.exec("INSERT INTO Test VALUES(1, 'Hello')", function (err,
affectedRows) {
  if (err) throw err;
  console.log('Number of affected rows:', affectedRows);
});
'''
```

Query

The exec function is a convenient way to completely retrieve the result of a query. In the below example all selected rows are fetched and returned in the callback:

```
conn.exec("SELECT * FROM Test WHERE ID < 5", function (err, rows) {
  if (err) throw err;
```

```
  console.log('Rows:', rows);
});
```

Values in the query can be substitued with JavaScript variables by using '?' placeholders in the query and passing an array of positional arguments.

```
conn.exec("SELECT * FROM Test WHERE ID BETWEEN ? AND ?", [5, 8],
  function (err, rows) {
  if (err) throw err;
  console.log('Rows:', rows);
});
```

Prepared Statement Execution

Prepared statements can be used for performance gain if the same statement is going to be executed many times. The following describes how to execute prepared statements and queries with the Node.js driver.

Prepare a Statement

The connection returns a statement object that can be executed multiple times.

```
conn.prepare('SELECT * FROM Test WHERE ID = ?', function (err,
stmt){
  if (err) throw err;
});
```

Execute a Statement

The execution of a prepared statement is similar to the direct statement execution. The first parameter of the exec function is an array with positional parameters.

```
stmt.exec([16], function(err, rows) {
  if (err) throw err;
  console.log("Rows: ", rows);
});
```

Execute a Batch Statement

The execution of a prepared batch statement is similar to the direct statement execution. The first parameter of the execBatch function is an array with positional parameters.

```
var stmt=conn.prepare("INSERT INTO Customers(ID, NAME)
VALUES(?, ?)");
stmt.execBatch([[1, 'Company 1'], [2, 'Company 2']], function(err,
rows) {
  if (err) throw err;
  console.log("Rows: ", rows);
});
```

Execute a Query

The execution of a prepared query is similar to direct statement execution. The first parameter of the execQuery function is an array with positional parameters.

```
var stmt=conn.prepare("SELECT * FROM Customers WHERE ID >= ? AND
ID < ?");
stmt.execQuery([100, 200], function(err, rs) {
  if (err) throw err;
    var rows = [];
    while (rs.next()) {
    rows.push(rs.getValues());
```

```
    }
  console.log("Rows: ", rows);
});
```

Drop a Statement

The below example drops a statement:

```
stmt.drop(function(err) {
  if (err) throw err;
});
```

Transaction Handling

Transactions are automatically committed.

Commit a Transaction

Setting auto commit to FALSE implicitly starts a new transaction that must be explicitly committed.

```
conn.setAutoCommit(FALSE);
conn.commit(function(err) {
  if (err) throw err;
  console.log('Transaction commited.');
});
```

Roll back a transaction

Setting auto commit to FALSE implicitly starts a new transaction that must be explicitly rolled back.

```
conn.setAutoCommit(FALSE);
conn.rollback(function(err) {
  if (err) throw err;
  console.log('Transaction rolled back.');
});
```

Related Information

Node.js Connection Properties [page 506]

SAP HANA Data Management Suite

4.4.1 Configure the Node.js Driver (Client Install)

Configure the Node.js driver so that you can use it to connect to an SAP HANA database.

Context

The Node.js driver is included with the SAP HANA client install and is also available for download from npmjs.com . The driver on the npm.sap.com may be newer than the one that is available with your SAP HANA client install.

Procedure

Install the driver.

Driver location	Description

Install the driver that is included with the client install	1. Run the following command:
	`npm install <folder>`
	`<folder>` is the path to the node subdirectory of your client install
	On UNIX, the default path is `npm install /usr/sap/hdbclient/node`.
	On Microsoft Windows, the default path is `npm install C:\Program Files\SAP\hdbclient\node`.
	2. The following code is included with your JavaScript application:
	`require('@sap/hana-client')`

Install the module from npmjs.com	Install the module: `npm install [-g] @sap/hana-client`
Deploy on Linux Alpine	Install libstdc++ as an APK package first, then install it as a dependency: `apk add --update --no-cache libstdc++`

Driver location	Description
	If libstdc++ is not installed then the following error is returned: `Error loading shared library libstdc++.so.6: No such file or directory (needed by hana-client.node)`

> **Note**
>
> Optionally, you can remove installed platform binaries using `HDB_NODE_PLATFORM_CLEAN` or `HDB_NODEJS_INSTALL_PLATFORMS`. See Environment Variables [page 20] for more information.

Example

The following example shows how to create a connection to the SAP HANA on-premise database and execute a query:

```
var hana = require('@sap/hana-client');
var conn = hana.createConnection();
var conn_params = {
  serverNode  : 'myserver:30015',
  uid         : 'system',
  pwd         : 'manager'
};
conn.connect(conn_params, function(err) {
  if (err) throw err;
  conn.exec('SELECT Name, Description FROM Products WHERE id = ?', [301],
function (err, result) {
    if (err) throw err;
    console.log('Name: ', result[0].Name, ', Description: ',
result[0].Description);
    // output --> Name: Tee Shirt, Description: V-neck
    conn.disconnect();
  })
});
```

4.4.2 Run the Node.js Examples

The SAP HANA client installation includes two example projects for Node.js.

Context

app1.js Opens a database connection, creates a table, inserts some data, and retrieves the inserted data.

app2.js Opens a database connection, creates a table, inserts some large LOB rows, and streams the LOB results via the createLobStream functionality.

Procedure

1. Open a Command Prompt (Microsoft Windows), bash shell (Linux), or Terminal (macOS).
2. Change the directory to the Node.js examples directory in the SAP HANA client installation.

   Microsoft Windows:

   ```
   cd "C:\Program Files\SAP\hdbclient\examples\node"
   ```

   Linux/macOS:

   ```
   cd /usr/sap/hdbclient/examples/node
   ```
3. Install the @sap/hana-cient module:

   ```
   npm install @sap/hana-client
   ```
4. Install the dependency modules:

   ```
   npm install async-waterfall
   ```
5. Change the example to provide the details for the host and port of the SAP HANA server, username, and password:

   ```
   var connOptions = {
             serverNode: 'myserver:30015',
             uid: 'User1',
             pwd: 'Password123'
           };
   ```
6. Run the example files:

   ```
   node app1.js
   node app2.js
   ```

4.4.3 Encrypting Node.js Connections

Use connection properties to encrypt Node.js connections.

> **Note**
>
> For Microsoft Windows, trusted certificate authorities can be managed through the Microsoft Management Console Certificates Snap-In. CommonCryptoLib is recommended and is the default if both are installed. CommonCryptoLib is installed separately from the SAP HANA client. On Microsoft Windows, the Microsoft Cryptography Provider is used by default and contains pre-authorized Certificate Authorities. You cannot use direct certificate strings with the Microsoft Cryptography Provider.

The following SAP HANA Cloud example uses a certificate string:

> **Note**
>
> This example does not work with Microsoft CryptoAPI (MSCrypto). For MSCrypto the Microsoft Windows Certificate Store is implicitly trusted. Remove the `CA` connection property in the example below to work with the SAP HANA Service or SAP HANA Cloud (Microsoft Windows already trusts the DigiCert Global Root G2).

```
var connOptions = {
     serverNode: '12345678-
abcd-12ab-34cd-1234abcd.hana.hanacloud.ondemand.com:443',
     uid: 'User1',
     pwd: 'Password123',
     encrypt: 'TRUE',
     ssltruststore: '-----BEGIN CERTIFICATE-----\n' +
'MIIDjjCCAnagAwIBAgIQAzrx5qcRqaC7KGSxHQn65TANBgkqhkiG9w0BAQsFADBh\n' +
'MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3\n' +
'd3cuZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBHbG9iYWwgUm9vdCBH\n' +
'MjAeFw0xMzA4MDExMjAwMDBaFw0zODAxMTUxMjAwMDBaMGExCzAJBgNVBAYTAlVT\n' +
'MRUwEwYDVQQKEwxEaWdpQ2VydCBJbmMxGTAXBgNVBAsTEHd3dy5kaWdpY2VydC5j\n' +
'b20xIDAeBgNVBAMTF0RpZ2lDZXJ0IEdsb2JhbCBSb290IEcyMIIBIjANBgkqhkiG\n' +
'9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuzfNNNx7a8myaJCtSnX/RrohCgiN9RlUyfuI\n' +
'2/Ou8jqJkTx65qsGGmvPrC3oXgkkRLpimn7Wo6h+4FR1IAWsULecYxpsMNzaHxmx\n' +
'1x7e/dfgy5SDN67sH0NO3Xss0r0upS/kqbitOtSZpLYl6ZtrAGCSYP9PIUkY92eQ\n' +
'q2EGnI/yuum06ZIya7XzV+hdG82MHauVBJVJ8zUtluNJbd134/tJS7SsVQepj5Wz\n' +
'tCO7TG1F8PapspUwtP1MVYwnSlcUfIKdzXOS0xZKBgyMUNGPHgm+F6HmIcr9g+UQ\n' +
'vIOlCsRnKPZzFBQ9RnbDhxSJITRNrw9FDKZJobq7nMWxM4MphQIDAQABo0IwQDAP\n' +
'BgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBhjAdBgNVHQ4EFgQUTiJUIBiV\n' +
'5uNu5g/6+rkS7QYXjzkwDQYJKoZIhvcNAQELBQADggEBAGBnKJRvDkhj6zHd6mcY\n' +
'1Yl9PMWLSn/pvtsrF9+wX3N3KjITOYFnQoQj8kVnNeyIv/iPsGEMNKSuIEyExtv4\n' +
'NeF22d+mQrvHRAiGfzZ0JFrabA0UWTW98kndth/Jsw1HKj2ZL7tcu7XUIOGZX1NG\n' +
'Fdtom/DzMNU+MeKNhJ7jitralj41E6Vf8PlwUHBHQRFXGU7Aj64GxJUTFy8bJZ91\n' +
'8rGOmaFvE7FBcf6IKshPECBV1/MUReXgRPTqh5Uykw7+U0b6LJ3/iyK5S9kJRaTe\n' +
'pLiaWN0bfVKfjllDiIGknibVb63dDcY3fe0Dkhvld1927jyNxF1WW6LZZm6zNTfl\n' +
'MrY=\n' +
'-----END CERTIFICATE-----';
};
var client = hdb.createClient(connOptions);
```

Related Information

SAP Note 3397584

SAP Note 3327214

SAP Note 3399572

Download and Install SAP Common Crypto Library

Node.js Connection Properties [page 506]

Server Certificate Authentication [page 28]

Configuring Clients for Secure Connections [page 28]

4.4.4 Node.js Connection Properties

When connecting to an SAP HANA database using node.js, you can specify node.js connection properties or SQLDBC connection properties. Property names are case insensitive.

> **Note**
>
> Do not set connection properties to an empty string. The client substitutes any empty string values with the property's default setting.

The following table lists the SAP HANA client-specific node.js connection properties.

Property	Value	Default	Description
`allowFetchWarnings`	Boolean	TRUE	Controls whether server warnings from retrieving result set of a query are returned to a JavaScript when it calls the next function of the ResultSet class. When this property is set to TRUE (default), the getWarnings function of the Connection class can then be used to retrieve warnings after the next function is called.
`ca`	`<string>`		Specifies the alias for `sslTrustStore`.
`cert`	`<string>`		Specifies the alias for `sslKeyStore`.
`charset`	`<character-set>`		Specifies the character set. This value must be either "ASCII", "UTF-8", or "UCS2".

Property	Value	Default	Description
`connectionLifetime`	`<number>`	0	Specifies the maximum time, in seconds, that the connection is cached in the implicit connection pool. A value of 0 causes implicit pooled connections to be cached permanently. When an implicit pooled connection is closed, its creation time is compared with the current time, and the connection is disrupted if that time span (in seconds) exceeds the value specified by `connectionLifetime`. Otherwise, the connection is returned to the implicit connection pool for future reuse.
`database` | `databaseName`	`<name>`		Specifies the name of the database to connect to.
`dataTruncationError`	Boolean	FALSE	When set to TRUE, fails the stored procedure call whenever a data truncation error occurs.
`host`	`<host-name>`		Specifies the name of the host to connect to.
`key`	`<string>`		Specifies the alias for `sslKeyStore`.
`maxPoolSize`	`<number>`	0	Defines the maximum number of connections that are allowed in the implicit connection pool. The default value is 0, meaning there is no limit.
`passphrase`	`<string>`		Specifies the alias for `sslKeyStorePassword`.

Property	Value	Default	Description
`pooling`	Boolean	FALSE	Specifies whether the Node.js driver opens a pooled connection. Implicit connection pools can improve performance for opening connections. Set pooling to TRUE to use implicit connection pools. For example: `var connOpts = { host: '<host-name>', port: <port>, uid: 'SYSTEM', pwd: 'manager', pooling: true } var conn = hanaClient.createClient(connOpts); conn.connect();`
`poolingCheck`	Boolean	FALSE	When using implicit connection pooling, this determines whether or not the pooled connection should be tested for viability before being reused.
`poolKey`	`<string>`		Specifies the key to use for the name of the implicit connection pool that this connection will be included in. Only applicable when pooling is TRUE.
`port`	`<port-number>`		Specifies the port number to connect to.
`pwd` | `password`	`<password>`		Specifies the password for the SAP HANA database user.

Property	Value	Default	Description
`resultSetArrayLimitMB`	`<number>` in MB	2048 (2 GB)	Limits the maximum amount of memory used for result set arrays returned by Connection.exec[ute] and Statement.exec[ute]. This can be helpful for applications that need to limit the maximum amount of memory used by the node process. The value is in MB and limits the approximate total amount of data in the result set before returning an error. The maximum value allowed is 2048 (2 GB). Specifying 0, any value greater than 2048, or a noninteger value defaults to 2048. **Note** The actual maximum amount of memory used by the various layers can be higher than the value of `resultSetArrayLimitMB`. For example, if `resultSetArrayLimitMB` is set to 10, the data for each result set is limited to approximately 10 MB, but a 10 MB result set could use 20 MB or more of peak memory during the node process.

Property	Value	Default	Description
`serverNode`	`<host-name>:<port>[, <host-name>:<port> ]`		Specifies the SAP HANA server address or addresses. If your system contains a fail-over standby node, you can specify it as an additional optional address. **Note** This property supports both `;` and `,` values only when you specify the Node.js connection.connect method with a hash of connection parameters first argument. These values do not work when you call the Node.js connection.connect method with a string-first argument. This property is interchangeable with the `host` and the `port` connection properties.
`spatialTypes`	Boolean	FALSE	Determines the native type returned by Statement.getColumnInfo and Statement.getParameterInfo for ST_POINT and ST_GEOMETRY columns or parameters. Possible values are: • 0 or FALSE returns 13 (VARBINARY) • 1 or TRUE - returns 75 (ST_POINT) and 74 (ST_GEOMETRY)
`uid|user`	`<user-ID>`		Specifies the name of an SAP HANA database user.

SQLDBC Connection Properties

The following table lists SQLDBC connection properties that can also be used with Node.js:

> **Note**
>
> A valid setting for a Boolean property can be: “1”, “TRUE”, “YES”, or “ON” for a TRUE setting or “0”, “FALSE”, “NO”, or “OFF” for a FALSE setting. The setting of a Boolean property is case-insensitive.

Property	Value	Default	Description
`abapVarCharMode`	Boolean	FALSE	Specifies that in ABAP VARCHAR mode, a single space and an empty string are considered the same.
`allowLocalCompress`	Boolean	TRUE	Determines whether or not connections from the same machine as the SAP HANA server are compressed. The `compress` parameter and/or the `indexserver.ini` > *session* > *compression* server configuration control whether or not network compression is enabled. **Note** If the connection uses a proxy, do not use `allowLocalCompress`=FALSE as the connection may not be compressed when required.
`allowReconnectOnSelect`	Boolean	OFF	Allows reconnects to happen for SELECT queries which fail during a receive. By default, this property is set to OFF, which allows for backward compatibility.

Property	Value	Default	Description
`bindAddress`	`<ip-address>`	NULL	Specifies that the provided address is used as the source address for the socket connection. By default, the address preferred by the OS is used. If setting `bindAddress` to a hostname, that hostname should only resolve to a single IP address. In the majority of network configurations, specifying `bindAddress` is not required. **Note** Only specify `bindAddress` if it is required to connect in your network configuration.
`chopBlanks`	Boolean	FALSE	Removes trailing blanks from character output.
`chopBlanksInput`	Boolean	FALSE	Removes trailing blanks from character input.
`communicationTimeout`	Timeout in milliseconds	0 (disabled)	Aborts communication attempts to the server after the specified timeout. If set to a non-zero value, the driver will wait for a maximum of the specified number of milliseconds for any request to the server to complete. If the time has expired and a response has not been received, the driver returns an error.

Property	Value	Default	Description
`compress`	Boolean	FALSE	Enables or disables network compression for the connection. The `indexserver.ini` > *session* > *compression* server configuration defaults to FALSE. Setting this parameter overrides the setting of the `indexserver.ini` > *session* > *compression* parameter. The `allowLocalCompress` property controls if clients running on the same machine as the server use network compression when compression is requested.
`connDownRollbackError`	Boolean	FALSE	Setting this property to TRUE specifies that if the database connection is down due to network or SAP HANA server problems, the client rolls back the last transaction and sends and error to the application. The application can resend the last transaction to the server after properly handling the error. The following conditions must be true: • The session is recoverable in the server • There is a written transaction (uncommitted operation) in the session • The client failed to send the last SQL statement, or the last statement was sent to the server successfully, but both auto-commit and DDL auto-commit in the client are off. By default, this property is set to FALSE.

Property	Value	Default	Description
`connectTimeout`	Timeout in milliseconds	0 (limited by the `nodeConnectTimeout` connection property)	Aborts connection attempts after the specified timeout. `connectTimeout` limits the overall time for the entire connection to try all server addresses specified. See `nodeConnectTimeout` for per-node timeout.
`currentSchema`	`<schema>`	CURRENTUSER	Sets the current schema, which is used for identifiers without a schema.
`cursorHoldabilityType`	NONE, COMMIT, ROLLBACK, COMMIT_AND_ROLLBACK	COMMIT	Specifies whether the cursor for a result set remains open after the transaction is committed or rolled back. If a cursor remains open, then the application is able to fetch more rows from it. NONE means the cursor is always closed after a commit or rollback, while COMMIT means the cursor remains valid after a commit, but not after rollback. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources. Support for holding a cursor over ROLLBACK or COMMIT_AND_ROLLBACK is available in SAP HANA servers beginning with SAP HANA 2.4.

Property	Value	Default	Description
`emptyTimestampIsNull`	Boolean	TRUE	When enabled, DATE, TIME, SECONDDATE, and TIMESTAMP values inserted as empty strings are returned as NULLs. When disabled, these values are returned as the following out-of-band values: • `DATE` - one day before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 00:00.000000000 UTC BC • `TIME` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `SECONDDATE` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `TIMESTAMP` - one hundred nanoseconds before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.999999900 UTC BC

Property	Value	Default	Description
`ignoreTopology`	Boolean	FALSE	Controls how the client uses the topology returned from the server. TRUE indicates to use the topology unless port-forwarding is detected and FALSE indicates to always ignore the topology. It is an error to specify both ignoreTopology=TRUE and siteType=PRIMARY or SECONDARY.
`isolationLevel`	TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE	TRANSACTION_READ_COMMITTED	Sets the isolation level for the connection.
`isoTimestampOutput`	`<string>`	FALSE	Displays whether or not 'T' is used as the separator between the Date and Time parts of a timestamp retrieved as a string: TRUE/FALSE. If FALSE, then a space character is used as the separator.
`key`	`<username>`	Empty	Specifies the key for the hdbuserstore.
`maxLazyDroppedStatements`	`<number>`	50	Controls the maximum number of queued statements to be dropped. Setting this value to 0 disables the lazy drop feature and causes a request to be sent to the server each time a statement is dropped by the application.
`maxVersion`	`<string>`		Specifies the maximum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.

Property	Value	Default	Description
`minVersion`	`<string>`		Specifies the minimum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.
`nodeConnectTimeout`	Timeout in milliseconds	60000	Aborts the connection attempt after the specified maximum timeout. The connection attempt typically fails immediately if the server cannot be found. If connection attempts to a particular server address are slow to fail, then using this connection property reduces the time before a successful connection by minimizing the time waiting to connect to individual addresses. On a host with multiple resolved IP addresses, the network timeout for each attempt to connect to an IP address will be equal to the `nodeConnectTimeout` divided by the number of resolved IP addresses. If this option is set to zero, the `connectTimeout` setting will be used. If both of the `nodeConnectTimeout` and `connectTimeout` are set to zero, then SQLDBC will use the system TCP/IP socket connect timeout to connect to the IP address.

Property	Value	Default	Description
`packetCaching`	Boolean	TRUE	Caching packets reduces the need to continually allocate and deallocate memory for packets. However, this increases the peak memory usage of the application, and for how long it stays at or near that peak.
`packetSize`	`<number>` in bytes	1048576 (1 MB)	Sets the maximum size of a request packet sent from the client to the server, in bytes. In some cases, request packets can be larger (for example, if a SQL statement or parameter value is larger than the packetSize). The minimum is 1 MB.
`packetSizeLimit`	`<number>` in bytes	2147483647 (2 GB)	Sets the maximum size for a request or reply packet, in bytes. If SQLDBC needs to allocate a packet larger than the limit, then it returns an error. If a reply packet is too large, then the connection is dropped. The minimum value is the packetSize value; the maximum value is 2 GB.

Property	Value	Default	Description
`prefetch`	Boolean	TRUE	Controls result set prefetching. When enabled, result set prefetch requests the next block of rows of a result set while the application is still processing the current block of rows. When fetching large result sets, result set prefetch can improve performance; however, it can also result in the client using more memory. In some cases, the server JobExecutor thread could be blocked from sending the reply for an indefinite period of time until the client receives the prefetch reply. If the result set contains LOB columns, then result set prefetch is always disabled.
`{ proxyHostname | proxy_host }`	`<host-name>`		Specifies the host name of the proxy server. The default proxy server is SOCKS5. If `proxyHttp` is set to true, the proxy server is assumed to be an HTTP proxy.
`proxyHttp`	Boolean	FALSE	When set to TRUE, enables HTTP proxy authentication and implies that ENCRYPT=TRUE. This property is only applicable if `proxyHostname` is specified and `webSocketURL` is not specified.

Property	Value	Default	Description
`{ proxyPassword | proxy_password }`	`<password>`		Specifies the password for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the Cloud Connector location ID value. The value must be entered as a base64 encoded value with a "1." prepended to the string.
`{ proxyPort | proxy_port }`	`<port-number>`		Specifies the port of the proxy server. The default is 1080.
`{ proxyScpAccount | proxy_scp_account }`	SAP BTP, Cloud Foundry Environment: `<location-ID>` SAP BTP, Neo Environment: `<subaccount>.<location-ID>`		Provides SAP Cloud Connector routing information to the Cloud Connector SOCKS proxy. In an SAP BTP, Cloud Foundry Environment, provide an optional `<locationID>` in plaintext. Internally, the value is converted to base64 encoding. This value overrides the `proxyPassword` connection property. In an SAP BTP, Neo Environment, either `proxyScpAccount` or `proxyUsername` must be specified. `<subaccount>` is mandatory and `<locationID>` is optional. Both values are entered as plaintext strings and are internally converted to base64 encoding. This value overrides the `proxyUsername` connection property.

Property	Value	Default	Description
`{ proxyUserName | proxy_userid }`	`<user-name>`		Specifies the user name for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the JSON Web Token (JWT) value. When using an SAP HANA Cloud Connector in the SAP BTP, Neo Environment, this field is used to provide the subaccount and location ID details. The values must be base64 encoded manually using the format `1.base64(subaccount).base64(locationID)`.
`reconnect`	Boolean	TRUE	Controls whether the system automatically reconnects to the database instance after a command timeout or when the connection is broken. Reconnecting restores the old state (for example, if no transaction was open).

Property	Value	Default	Description
`resolveHostName`	IPv4First IPv6First	IPv4First	Controls which addresses to use in a server address, especially if an address or hostname resolves to multiple IP addresses. In all cases, stop attempting further resolved addresses once a TCP/IP connection is successfully established. The following values can be used with `resolveHostName`: • IPv4First (default) - try all resolved addresses starting with IPv4 addresses. • IPv6First - try all resolved addresses starting with IPv6 addresses.
`sessionVariable:<key-name>`	`<session variable value>`	None except for APPLICATION and APPLICATIONUSER keys	Sets the session variable `<key-name>` to `<session variable value>`. For example, to set the session variable with key `APPLICATION`, use `sessionVariable:APPLICATION=MyApplication`.

Property	Value	Default	Description
`statementCacheSize`	`<number>`	0 (no statements cached)	Controls the maximum number of cached prepared statements. Caching SQL statements that are frequently prepared and dropped can improve performance. DDL can cause behavior changes when statement caching is enabled. **Note** If it is possible that the definition of objects referred to by the prepared statements may change because DDL is executed while you are connected, then it is recommended that you do not enable prepared statement caching.
`statementRoutingFailureBackoff`	Boolean	TRUE	Enables or disables the re-establishment of an internal connection to a specific host for a time period, after the internal connection fails during the preparation or execution of a statement routed to that host. The minimum time period is 1 minute. The time period is doubled after each subsequent internal connection establishment or usage failure for the same node, up to a maximum of 1 hour. Every successful internal connection re-establishment and usage resets the time period to the minimum value.

Property	Value	Default	Description
`statementRoutingWarnings`	Boolean	FALSE	Returns a SQL warning to the application with SQLCODE -11001 or -11003 that gives information about the routing failure of a prepared statement.
`tcpKeepAliveCount`	`<number-of-retries>` This value is ignored if it is 0.	5	Controls the number of KeepAlive packets sent to the server before the driver assumes that the connection is closed. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.
`tcpKeepAliveIdle`	`<seconds>` This value is ignored if it is 0.	200	Controls the number of seconds that the connection is idle before the driver sends a KeepAlive packet. If 0 is specified, this value is determined by the operating system.
`tcpKeepAliveInterval`	`<seconds>` This value is ignored if it is 0.	4	Controls the number of specified seconds before the driver sends an additional KeepAlive packet if there was no response to the previous KeepAlive packet. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.

Property	Value	Default	Description
`timestampPadding`	Boolean	TRUE	In a timestamp where the format contains a fraction, this property determines whether the fractional portion of the timestamp will be 7 or 9 digits. When disabled, the fractional portion is 7 digits to match how the server displays timestamps. Otherwise, it is 9 digits. **Note** ABAP timestamps are not affected regardless of the value of `timestampPadding`.
`traceFile`			Specifies the trace file name. Only the stdout and stderr values are supported.
`traceOptions`			Implicitly enables tracing for all SQLDBC-based client connections. See the HDB_SQLDBC_TRACEOPTS description in the **Environment Variables** topic for more information.
`webSocketURL` (SAP HANA Service only)	`<webSocketUrl>`		Specifies the WebSocket URL used to connect to a SAP HANA server. The connection property value cannot be empty. WebSockets connections have the following default connection property settings: • encrypt=TRUE • compress=TRUE • reconnect=FALSE

Security Connection Properties

Property	Value	Default	Description
`{ cseKeyStorePassword | clientside_encryption_keystore_password }`	`<string>`		Provides the password for the local key store. This option is required when using client-side encryption. Passwords must be at least eight characters and have at least one upper case character, one lower case character, and one number.
`authenticationMethods`	`<comma-separated-string>`		Provides a comma-separated list of authentication methods to be used by the client. Possible values are: password, pbkdf2, ldap, saml, jwt, saplogon, kerberos, sessioncookie, and X509.
`authenticationX509`	`<string>`	`<value-of-sslKeyStore>`	Provides a string containing the X509 certificate or the name of a file containing the X509 certificate. The certificate must be PEM-encoded in PKCS8 format. Defaults to the existing value of `sslKeyStore`, if specified.
`authenticationX509Password`	`<password>`		With CommonCryptoLib, specifies the PIN for the PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`encrypt`	Boolean	FALSE	Enables or disables TLS encryption. For SAP HANA versions 2.5 and lower, set encrypt=TRUE when connecting to port 443 (SAP HANA Cloud). For SAP HANA versions 2.6 and higher, encrypt=TRUE is not required when connecting to port 443.

Property	Value	Default	Description
`sslCryptoProvider`	{ commoncrypto | openssl | mscrypto }	1. commoncrypto (if installed) 2. openssl/mscrypto	Specifies the cryptographic library provider used for TLS communication. If you specify a value for this property, then you must also explicitly specify paths in both the `sslKeyStore` and `sslTrustStore` properties to avoid configuration issues. If CommonCryptoLib is not available, OpenSSL is used by default in Linux environments, and msCrypto in Microsoft Windows environments. OpenSSL is not available on Microsoft Windows. **Note** Check the `client` folder of the installation package to see if `COMMONCRYPTOLIB.TGZ` is present. If not, you can download CommonCryptoLib separately. For instructions on how to download CommonCryptoLib, see "Download and Install SAP Common Crypto Library" in the *SAP HANA Client Installation and Update Guide*.
`sslHostNameInCertificate`	`<string>`	Empty	Specifies the host name used to verify server's identity. The host name specified here verifies the identity of the server instead of the host name with which the connection was established. For example, in a single-host system, if a connection is established from a client on the same host as the server, then a mismatch would arise between the host named in the certificate (actual host name) and the host used to establish the connection (localhost). If you specify `*` as the host name, then the server's host name is not validated. Other wildcards are not permitted.

Property	Value	Default	Description
`sslKeyStore`	`<file>` | `<PEM-encoded-identity>` msCrypto: MY | Root | Trust | CA	`$SECUDIR/sapcli.pse` (CommonCryptoLib) `$HOME/.ssl/key.pem` (OpenSSL) MY (msCrypto)	Specifies the path to the keystore file that contains the client's identity. An identity is used for mutual TLS authentication and consists of the client's private key, the client's certificate, and, optionally, the certificate of the signing authority that signed the client's certificate. If you are using CommonCryptoLib, then the PSE file must also contain the trust store (for example, the server's public certificates) and the sslTrustStore property should be empty. If you are using CommonCryptoLib, but not doing mutual TLS authentication, then the PSE file contains only the trust store. If you are using CommonCryptoLib, then use the SAPGENPSE tool (installed with CommonCryptoLib) to create the client PSE (`sapcli.pse`) and generate the client's keys. You must also import the server's public certificates into `sapcli.pse`. Typically, this is the root certificate or the certificate of the certification authority that signed the server's public certificates. See *SAP Note 1718944 (SAP HANA DB: Securing External SQL Communication (CommonCryptoLib))*. If you are using OpenSSL, use OpenSSL tools to create the required keystore file. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.

Property	Value	Default	Description
`sslMinProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	TLS12	Controls the minimum TLS version. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The lowest protocol version is accepted in TLS handshakes.
`sslMaxProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	MAX	Controls the maximum TLS version. MAX allows this parameter to be future-proof in the event that new TLS versions are supported. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The highest negotiated protocol version is used in TLS handshakes.
`sslKeyStorePassword`	`<password>`		With CommonCryptoLib, specifies the PIN for the keystore PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`sslSNIHostname`	`<string>`	Empty	Specifies the name of the host that is attempting to connect at the start of the TLS handshaking process.
`sslSNIRequest`	Boolean	TRUE	Specifies whether SNI requests are enabled for TLS connections: TRUE/FALSE.

Property	Value	Default	Description
`sslTrustStore`	`<file>` | `<PEM-encoded-trust-store>` msCrypto: MY | Root | Trust | CA	`$HOME/.ssl/trust.pem` (OpenSSL) sapcli.pse (CommonCryptoLib) MY (msCrypto)	Specifies the path to a trust store file that contains the server's public certificates if using OpenSSL. Typically, the trust store contains the root certificate or the certificate of the certification authority that signed the server's public certificates. If you are using the cryptographic library CommonCryptoLib or msCrypto, leave this property empty. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.
`sslUseDefaultTrustStore`	Boolean	TRUE	(Only applies to the OpenSSL Crypto Provider) Set to FALSE in order to disable the use of the default Certificate Authorities provided with OpenSSL or the Operating System.
`sslValidateCertificate`	Boolean	TRUE	Specifies whether to validate the server's certificate.

4.4.5 Node.js Connection Pooling

Connection pooling allows applications to reuse existing connections by automatically saving the connection to a pool so it can be reused, rather than repeatedly creating a new connection to the SAP HANA database server.

There are an unlimited number of open connections inside and outside of the pool. The maximum can be set with the environment variable HDB_NODEJS_MAX_OPEN_CONNECTIONS.

The number of open connections inside and outside the pool can be determined by using `getOpenConnectionCount()`.

The node.js driver supports two different forms of connection pooling: implicit connection pooling and the explicit connection pooling API. Implicit connection pooling uses the `pooling`=true connection property to enable pooling of connections with the same connection string. The explicit connection pooling API uses the hana.createPool method to create each connection pool. If an application uses connection pooling, it must always use one or the other form of connection pooling. Attempting to use both forms of connection pooling is not allowed and results in errors.

Related Information

Implicit Connection Pooling [page 531]
Explicit Connection Pooling API [page 532]
Node.js Connection Properties [page 506]
ConnectionPool Class [page 554]
Hana Class [page 561]

4.4.5.1 Implicit Connection Pooling

Enable pooling of connections based on either a connection string or pool key.

There are sub-pools in the pool, with sub-pools sharing connections with an empty pool key and the same connection string or with the same pool key. The maximum number of connections in a sub-pool can be set using the `maxPoolSize` connection property.

Implicit connection pooling is enabled and disabled by using the `pooling` connection property. By default, implicit connection pooling is disabled and must be enabled by specifying `pooling`=true.

The size of the implicit pool can be set by using the setConnectionPoolLimit(Integer) Method [page 566].

The maximum pool size for a specific connection string or option is set by using the `maxPoolSize` connection property. The `maxPoolSize` default value is 0, meaning that there is no limit.

The global maximum pool size is set by using the setConnectionPoolLimit(Integer) Method [page 566]. The default value of 0 does not set a limit on the number of connections in the implicit connection pool.

The implicit connection pool can be emptied by using the clearPool([Function]) Method [page 540].

When your application first attempts to connect to the SAP HANA database server, it checks the pool for an existing connection that uses the same connection properties you specified. If a matching connection is found, then that connection is used. Otherwise, a new connection is used.

When you disconnect, the connection is cleaned up and returned to the pool so that it can be reused. However, the connection does not return to the pool if the pool size maximum set with the setConnectionPoolLimit(Integer) Method [page 566] and/or the sub-pool size maximum set with the `maxPoolSize` connection property has been reached. If the connection being disconnected is not put back into the pool, it is disconnected from the server.

If the `poolingCheck` connection property is set to true, the Node.js driver specifies that connections in the connection pool should be tested for viability before being reused. Connection viability is tested on all physical connections with the dummy SQL statement `SELECT 'ping' FROM DUMMY`. Connections deemed viable are reused; connections deemed not viable are not used and are removed from the implicit connection pool.

Sub-pools can be named if the `poolKey` connection property is specified. Named sub-pools can contain connections with differing usernames and/or passwords. When a connection is retrieved from the pool, it will first check to see if a connection with matching username and password is already in the pool, in which case it will use that as-is. If no matching connection is found in the pool, it will retrieve a connection with a differing username and/or password and use a CONNECT SQL statement to set its username and password before retrieving it from the pool. Only connections using basic authentication or JWT currently support the `poolKey` property.

Example

The following on-premise example uses implicit connection pooling to connect to an SAP HANA database server:

```
server=hana-db:30015;pooling=true;maxPoolSize=50;userid=JSmith;password=secret
```

Related Information

Explicit Connection Pooling API [page 532]
Node.js Connection Properties [page 506]
Connection Class [page 536]

4.4.5.2 Explicit Connection Pooling API

Create multiple customized connection pools with explicit connection pooling.

Other features that differentiate explicit connection pooling from implicit connection pooling are:

- Create pools that support different users
- Permission to directly obtain a connection from the pool
- Set the capacity of the connection pool
- Specify a time-out functionality to evict connections from the pool

To create an explicit connecton pool and to see a list of explicit connection pool properties, see the createPool({String | Object}[, Object]) Method [page 563].

> ⚠ Caution
>
> When using explicit connection pooling, do not specify the following Node.js connection properties:
>
> - `pooling`
> - `maxPoolSize`
> - `connectionLifetime`
> - `poolingCheck`

> ⓘ Note
>
> When the connections to the server are closed, the server does not drop temporary tables.

Related Information

ConnectionPool Class [page 554]

Hana Class [page 561]
Implicit Connection Pooling [page 531]
Node.js Connection Properties [page 506]

4.4.5.3 Cleaning A Connection Before Reuse

When a connection is disconnected by either implicit and explicit connection pooling, some cleaning is performed before returning it to the pool.

The cleaning performed before returning a connection to the pool includes:

- Rolling back the transaction
- Unsetting all session variables
- Setting the isolation level (the Node.js driver sets the isolation level to the default value, read committed, or the value provided by the user)
- Resetting the locale when a pooled connection is returned to the connection pool if the user provided the locale with the connection string or connection properties
- Setting the schema if it is provided by the user

The Node.js driver does not drop temporary tables when a pooled connection is cleaned before reuse.

Related Information

Implicit Connection Pooling [page 531]
Explicit Connection Pooling API [page 532]
clearPool([Function]) Method [page 540]

4.4.6 Node.js Tracing

Use tracing to help diagnose errors in JavaScript applications that use the Node.js driver.

There are several ways to activate Node.js tracing:

- Tracing configured by hdbsqldbc_cons
- Tracing environment variables
- Connection-specific tracing to the console
- Tracing using a callback provided by the Node.js application

Trace Using the hdbsqldbc_cons Tool

Use the hdbsqldbc_cons tool to initiate diagnostic tracing for your Node.js applications. The API category of tracing can be particularly helpful for tracing calls into the Node.js driver.

Example

```
hdbsqldbc_cons CONFIG TRACE FILENAME /tmp/NODEJS-%p.txt
hdbsqldbc_cons TRACE API ON LEVEL DEBUG
hdbsqldbc_cons TRACE SQL ON LEVEL INFO
```

See hdbsqldbc_cons Trace Options [page 789] for more information.

Trace Using Environment Variables

There are two environment variables that can be used to configure tracing for Node.js applications: HDB_SQLDBC_TRACEFILE and HDB_SQLDBC_TRACEOPTS.

HDB_SQLDBC_TRACEFILE	Specifies the trace file name. A timestamp is added to the name to provide additional protection from malicious users.
HDB_SQLDBC_TRACEOPTS	Specifies the trace configuration options. The API trace category can be particularly helpful for tracing calls into the Node.js driver.

Example

```
set HDB_SQLDBC_TRACEFILE="/tmp/NODEJS-%p.txt"
set HDB_SQLDBC_TRACEOPTS="API=DEBUG,SQL=INFO"
```

See SQLDBC Trace Options [page 786] for more information.

Connection-Specific Tracing to the Console

There are two connection options that can be used to configure tracing for specific Node.js connections to the console: traceFile and traceOptions.

traceFile	Specifies the trace file name. Only the stdout and stderr values are supported.
traceOptions	Specifies the trace configuration options. The API trace category can be particularly helpful for tracing calls into the Node.js driver.

Example

```
var connOptions = {
    serverNode: 'myserver:30015',
    uid: 'User1',
```

```
    pwd: 'Password123',
    traceFile: 'stdout',
    traceOptions: 'API=DEBUG,SQL=INFO'
};
```

See SQLDBC Trace Options [page 786] for more information.

Trace Using a Node.js Callback Method

A Node.js trace callback method can be supplied to a connection using the onTrace method to receive trace messages.

Example

```
var traceCB = function (buf) {
    console.log(buf);
};
conn.onTrace("API=DEBUG,SQL=INFO,FLUSH,OutBufferSize=64k", traceCB);
```

See onTrace(traceOptions, callback) Method [page 549] for more information.

4.4.7 Node.js TypeScript Support

TypeScript is a syntactical superset of the JavaScript programming language that adds support for optional static typing. To take advantage of this static typing with the SAP HANA Node.js driver, the type declaration file `index.d.ts` is included within the driver installation.

Using the SAP HANA Node.js driver with TypeScript requires the `@types/node` package which can be installed as follows:

```
npm install @types/node
```

Example

The following example demonstrates a simple statement prepare and execute using TypeScript:

```
import * as hdb from '@sap/hana-client';
var conn_params: hdb.ConnectionOptions = {
    serverNode : 'myserver:30015',
    uid        : 'system',
    pwd        : 'manager'
};
var conn: hdb.Connection = hdb.createConnection();
conn.connect(conn_params, function(err: Error) {
    if (err) throw err;
    conn.prepare("SELECT TO_INTEGER(?) AS ID, ? AS VALUE FROM DUMMY",
function(err: Error, stmt: hdb.Statement) {
        if (err) throw err;
```

```
        interface TestResultRow {
            ID: number,
            VALUE: string
        }
        // The execute method itself cannot know the type of the result, so this
is passed in as a type variable (TestResultRow[])
        stmt.execute<TestResultRow[]>([1, 'TestValue'], function(err: Error,
res: TestResultRow[]) {
            if (err) throw err;
            console.log('ID: ', res[0].ID, ", VALUE: ", res[0].VALUE);
            conn.disconnect();
        });
    });
});
```

4.4.8 Connection Class

Represents the connection to the database.

Syntax

```
class Connection
```

Members

All members of the connection.

Methods

Type	Method	Description
	abort([Function]) Method [page 539]	Aborts the running database request that is being executed on the connection.
	clearPool([Function]) [page 540]	Empties the connection pool.
	commit([Function]) [page 541]	Performs a commit on the connection.
	connect({String | Object} [, Function]) [page 542]	Creates a new connection to the database by using a connection string or a hash of connection parameters that is passed in as a parameter.
	{disconnect | close}([Function]) [page 543]	Closes the current connection.
Result, Number, or Undefined.	exec[ute](String[, Array][,Object][, Function]) Method [page 544]	Executes the specified SQL statement.

Type	Method	Description
	getClientInfo(String) [page 547]	Returns the string of the specified property for the connection.
	getWarnings() Method [page 548]	Returns an array of warnings for the connection.
	getOpenConnectionCount() Method [page 548]	Returns the current number of open connections.
	onTrace(traceOptions, callback) Method [page 549]	Provides a callback to receive trace messages.
Statement	prepare(String[, Function]) [page 550]	Prepares the specified SQL statement and returns a Statement object.
	rollback([Function]) [page 551]	Performs a rollback on the connection.
	setAutoCommit(Boolean) [page 552]	Changes the autocommit setting for the connection.
	setClientInfo(String, String) [page 552]	Sets a client information property for the connection.
	setRowSetSize (Integer) Method [page 553]	Specifies the size of the row set.
	state() Method [page 554]	Returns a string that indicates the state of the connection.

Remarks

Once the connection is made, if there are network issues, any call using the connection fails and returns an error.

Example

The following example uses synchronous calls to create a new connection to the SAP HANA on-premise database server, issue a SQL query against the server, display the result set, and then disconnect from the server.

```
var hana=require('@sap/hana-client');
var client=hana.createConnection();
client.connect({serverNode:'myserver:30015',uid:'system',pwd:'manager'});
console.log('Connected');
result = client.exec("SELECT * FROM Customers");
console.log(result);
client.disconnect();
```

```
console.log('Disconnected');
```

The following on-premise example performs the same tasks as the previous example by using callbacks to perform asynchronous calls. Error checking is included.

```
var hana = require('@sap/hana-client');
var client = hana.createConnection();
client.connect("serverNode=myServer:30015;uid=MyUser;pwd=MyPassword",
  function(err)
  {
    if(err) {
      console.error("Connect error: ",err);
    } else {
      console.log("Connected");

      client.exec("SELECT * FROM Customers",
        function(err,rows)
        {
          if(err) {
            console.error("Error: ",err);
          } else {
            console.log(rows);
          }
        }
      );

      client.disconnect(
        function(err)
        {
          if(err) {
            console.error("Disconnect error: ", err);
          } else {
            console.log("Disconnected");
          }
        }
      );
    }
  }
);
```

The following on-premise example also uses callbacks, but the functions are not inlined.

```
var hana=require('@sap/hana-client');
var client=hana.createConnection();
client.connect("serverNode=myServer:30015;uid=myUser;pwd=myPassword",
async_connect);
function async_connect(err)
{
  if(err) {
    console.error("Connect error: ", err);
  } else {
    console.log("Connected");
    client.exec("SELECT FROM Customers",async_results);
  }
}
function async_results(err,rows)
{
  if(err) {
    console.error("Error: ", err);
  } else {
    console.log(rows);
  }
  client.disconnect(async_disconnect);
}
function async_disconnect(err)
{
```

```
  if(err) {
    console.error("Disconnect error: ", err);
  } else {
    console.log("Disconnected");
  }
}
```

You can also pass connection parameters into the createConnection function. Those connection parameters are combined with connection parameters in the connect() function call to get the connection string used for the connection. Use a hash of connection parameters or a connection string fragment in either call using the following on-premise example:

```
var hana=require('@sap/hana-client');
var client=hana.createConnection({uid: 'myUser'; pwd: 'myPassword'});
client.connect('serverNode=myServer:30015');
// the connection string that is used is
// "uid=myUser;pwd=myPassword;serverNode=myServer:30015"
```

4.4.8.1 abort([Function]) Method

Aborts the running database request that is being executed on the connection.

Syntax

```
connection.abort([callback])
```

Parameters

Type	Name	Description
Function	`callback`	The optional callback function.

Remarks

This method does nothing if there is no running request on the connection. If there is a running request on the connection, the client aborts waiting for the reply and then drops the connection to the server. Future method calls on the aborted and dropped connection may result in the connection being reconnected.

4.4.8.2 clearPool([Function]) Method

Empties the connection pool.

Syntax

```
connection.clearPool([callback] )
```

Parameters

Type	Name	Description
Function	`callback`	Specifies the optional callback function.

Returns

The number of connections in the pool.

Remarks

By default, the Node.js driver does not have implicit connection pooling enabled. Enable connection pooling by setting the `<pooling>` connection property to true.

The clearPool() Method fully disconnects and removes pooled connections from the connection's implicit connection pool. All implicity pooled connections are removed independent of the current connection's connection string.

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err){
    };
```

Example

The following synchronous example shows how to use the `clearPool`() method:

```
var conn = hdb.createClient(connStr);
var count = conn.clearPool();
```

Related Information

Implicit Connection Pooling [page 531]
Cleaning A Connection Before Reuse [page 533]

4.4.8.3 commit([Function]) Method

Performs a commit on the connection.

Syntax

```
connection.commit([callback])
```

Parameters

Type	Name	Description
Function	`callback`	The optional callback function.

Remarks

This method performs a commit on the connection if autoCommit is set to false. By default, inserts, updates, and deletes are committed when statements are executed.

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err){
    };
```

Example

The following example shows how to use the commit method synchronously.

```
var hana=require('@sap/hana-client');
    var client=hana.createConnection();
    client.connect("serverNode=myserver:30015;uid=system;pwd=manager");
    client.setAutoCommit(false);
    stmt=client.prepare(
    "INSERT INTO Departments "
    + "(DepartmentID,DepartmentName, DepartmentHeadID)"
    + "VALUES(?,?,?)");
```

```
    var result=stmt.exec([600, 'Eastern Sales', 902]);
    result+=stmt.exec([700, 'Western Sales', 902]);
    stmt.drop();
    console.log("Number of rows added: " + result);
    result=client.exec("SELECT * FROM Departments");
    console.log(result);
    client.commit();
    client.disconnect();
```

4.4.8.4 connect({String | Object}[, Function]) Method

Creates a new connection to the database by using a connection string or a hash of connection parameters that is passed in as a parameter.

Syntax

```
connection.connect(<connection_parameters>[, <callback>])
```

Parameters

Type	Name	Description
String or Object	`<connection_parameters>`	A valid connection string or a hash of connection parameters.
Function	`<callback>`	The optional callback function.

Remarks

Before the end of the program, call the disconnect method to free up resources.

The charset (CS) connection parameter CS=UTF-8 is always appended to the end of the connection string by the driver since all strings must be sent in that encoding.

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err)
    {
    };
```

Example

The following example shows how to use the connect method synchronously.

```
var hana=require('@sap/hana-client');
var conn=hana.createConnection();
client.connect("serverNode=myserver:30015;uid=system;pwd=manager");
```

Related Information

{disconnect | close | end}([Function]) Method [page 543]

4.4.8.5 {disconnect | close | end}([Function]) Method

Closes the current connection.

Syntax

```
connection.{disconnect | close | end}([callback])
```

Parameters

Type	Name	Description
Function	`callback`	The optional callback function.

Remarks

Call this method before the program ends to free up resources.

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err)
    {
    };
```

Example

The following examples shows how to use the disconnect method synchronously.

```
var hana=require('@sap/hana-client');
    var client=hana.createConnection();
    client.connect("serverNode=myserver:30015;uid=system;pwd=manager");
    client.disconnect();
```

Related Information

connect({String | Object}[, Function]) Method [page 542]

4.4.8.6 exec[ute](String[, Array][,Object][, Function]) Method

Executes the specified SQL statement.

Syntax

```
connection.exec[ute](sql[, parameters] [, options] [, callback])
```

Parameters

Type	Name	Description
String	`sql`	Specifies the SQL statement to execute.
Array	`parameters`	Specifies an optional array of bind parameters.
Object	`options`	Specifies the optional object containing the options for defining the representation of rows in the result. Option keys can be `nestTables` and `rowsAsArray`.
Function	`callback`	Specifies the optional callback function.

For queries that produce result sets, this method returns an array of row objects. The **Options** field defines the representation of each row, along with options applicable to the query as a whole. The following options can be used:

- The default row representation is an object where each property name is the column name and the property value is the column value. If multiple columns have the same column name, the property name

for the second and subsequent columns with the same column name is "<column-name>:<n>" where <n> is a number starting from 1.

- {rowsAsArray: true} makes the row representation an array of column values. Using this format may result in improved performance.
- {nestTables: true} makes the row representation a nested object. If the table names for columns are known, the object property names are the table names. The object property values are themselves objects, where each property name is the column name, and the property value is the column value. If the table names for columns are not known, then the representation is the same as the default representation.
- {queryTimeout: int} ensures that the server will time out within the specified time (in seconds) when a connection is queried.
- {communicationTimeout: int} ensures that the client will time out within the specified time, in milliseconds, when a connection is queried.

Returns

Returns the number of rows affected and undefined.

If no callback is specified, then the result is returned.

> **Note**
>
> This method returns the first result set. To return more than one result set use exec[ute]Query([Array][, Object][, Function]) Method [page 590]. To move from one result set to the next use nextResult([Function]) Method [page 583].

Remarks

This method takes in an SQL statement and an optional array of bind parameters to execute.

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err, result)
{
};
```

For queries that produce result sets, the result set object is returned as the second parameter of the callback function.

> **Note**
>
> For queries, the entire result set is returned as an array of rows. Since large result sets can use significant memory and time, it is recommended that a WHERE or LIMIT clause is used where possible. The `rowsAsArray` option is recommended for best performance.

For INSERT, UPDATE, and DELETE statements, the number of rows affected is returned as the second parameter of the callback. For other statements, the result is undefined.

Example

The following synchronous example shows how to use the exec method:

```
var hana=require('@sap/hana-client');
var client=hana.createConnection();
client.connect("serverNode=myserver:30015;uid=system;pwd=manager");
result=client.exec("SELECT * FROM Customers");
console.log(result);
client.disconnect();
```

The following synchronous example shows how to specify bind parameters:

```
var hana=require('@sap/hana-client');
var client=hana.createConnection();
client.connect("serverNode=myserver:30015;uid=system;pwd=manager");
result=client.exec(
"SELECT * FROM Customers WHERE ID >=? AND ID <?",
[300, 400]);
console.log(result);
client.disconnect();
```

To return a separate object per row for each table when you specify a SQL statement as a join with overlapping column names, set the `nestTables` option to true. For example:

```
var command = 'SELECT top 1 * FROM t1 JOIN t2 on t1.id = t2.id';
var options = {nestTables: true};
connection.exec(command, options, function(err, rows)
{// process rows
});
```

Specifying the above example returns an array like the following example:

```
[{
  T1: {
   ID: 1,
   A: 't1.1.a',
   B: 't1.1.b'
  },
  T2: {
   ID: 1
   A: 't2.1.a',
   B: 't2.1.b'
  }
}]
```

To return all rows as an array where the order of the column values is exactly the same as in Statement.getColumnInfo, set the `rowsAsArray` option to true. For example:

```
var command = 'SELECT top 1 * FROM t1 JOIN t2 on t1.id = t2.id';
connection.exec(command, {rowsAsArray: true}, function(err, rows) { /* process
rows */ });
```

Specifying the above example returns an array like the following example:

```
[[
  1,
  't1.1.a',
  't1.1.b',
  1,
  't2.1.a',
```

```
  't2.1.b'
]]
```

4.4.8.7 getClientInfo(String) Method

Retrieves the value of the specified property for the connection.

Syntax

```
connection.getClientInfo(key)
```

Parameters

Type	Name	Description
String	key	The property whose value you are requesting.

Returns

Returns a string that contains the value of the specified property.

Remarks

The supported properties include the session-specific client variables (APPLICATION, APPLICATIONUSER, CLIENT, and USER), user-defined variables, and the properties listed in the M_SESSION_CONTEXT system view.

Related Information

Setting Session-Specific Client Information [page 16]

4.4.8.8 getOpenConnectionCount() Method

Returns the current number of open connections.

Syntax

```
connection.getOpenConnectionCount()
```

Returns

Returns the number of open connections in the form of an integer.

Remarks

Use this method to track the number of open connections over time to ensure that the number of connections are not endlessly growing, causing unnecessary memory usage.

4.4.8.9 getWarnings() Method

Returns an array of warnings for the connection.

Syntax

```
connection.getWarnings()
```

Returns

Returns an array of warnings. Each warning is an object with code, message, and sqlState properties.

Remarks

Each connection stores warnings that are generated by database operations. Use this method to get the warnings generated for the current connection. This method is a non-blocking method and does not support a callback. After you call this function, the connection clears all warnings.

4.4.8.10 onTrace(traceOptions, callback) Method

Provides a callback to receive trace messages.

Syntax

```
connection.onTrace(traceOptions, callback)
```

Parameters

Type	Name	Description
String	`traceOptions`	A string list of trace options.
Function	`callback`	The callback function.

Remarks

The trace option supported by onTrace is `OutBufferSize`, which specifies the size of the buffer to use to accumulate trace messages for the callback. See SQLDBC Trace Options [page 786] for more information.

The callback takes a single parameter, a string object containing a null-terminated block of trace messages, each one separated by a new line:

```
var traceCallback = function (buf) { ... }
```

The main thread will call this callback periodically, passing it all the trace messages that have accumulated for this connection. If more trace messages than can fit in the buffer have been generated since the last time the callback was called, the extra messages will be discarded and a trace message will be written indicating that trace lines have been skipped. To avoid losing trace messages, specify a larger `OutBufferSize` or use reduced trace setting levels in a node application.

It is recommended to call the onTrace method before calling connection.connect() so that the connection attempt can be traced. However, it can be called at any time on a valid connection object.

Example

The following example enables DEBUG and SQL tracing and writes the trace out to the console:

```
var connOptions = {
    serverNode: 'myserver:30015',
    uid: 'User1',
    pwd: 'Password123'
};
var conn = hdb.createConnection();
```

```
var traceCB = function (buf) {
    console.log(buf);
};
conn.onTrace("DEBUG,SQL,FLUSH,OutBufferSize=64k", traceCB);
conn.connect(connOptions, function(err) {
    conn.exec('select 1 from dummy', function(err) {
        conn.disconnect();
    });
});
```

Related Information

SQLDBC Tracing [page 785]

4.4.8.11 prepare(String[, Function]) Method

Prepares the specified SQL statement and returns a Statement object.

Syntax

```
connection.prepare(sql[, callback])
```

Parameters

Type	Name	Description
String	`sql`	The SQL statement to be prepared.
Function	`callback`	The optional callback function.

Returns

If no callback is specified, then a Statement object is returned.

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err, Statement)
```

```
    {
    };
```

Example

The following synchronous example shows how to use the prepare method.

```
var hana=require('@sap/hana-client');
    var client=hana.createConnection();
    client.connect("serverNode=myserver:30015;uid=system;pwd=manager");
    stmt=client.prepare("SELECT * FROM Customers WHERE ID >= ? AND ID < ?");
    var result=stmt.exec([200, 300]);
    console.log(result);
    client.disconnect();
```

4.4.8.12 rollback([Function]) Method

Performs a rollback on the connection.

Syntax

```
connection.rollback([callback])
```

Parameters

Type	Name	Description
Function	`callback`	The optional callback function.

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err){
    };
```

Example

The following synchronous example shows how to use the rollback method.

```
var hana=require('@sap/hana-client');
    var client=hana.createConnection();
    client.setAutoCommit(false);
    client.connect("serverNode=myserver:30015;uid=system;pwd=manager");
    stmt=client.prepare(
    "INSERT INTO Departments"
    +"(DepartmentID, DepartmentName, DepartmentHeadID)"
    +"VALUES(?,?,?)");
    var result=stmt.exec([600, 'Eastern Sales',902]);
    result+=stmt.exec([700, 'Western Sales', 902]);
    stmt.drop();
    console.log("Number of rows added: " + result);
    result=client.exec("SELECT * FROM Departments");
    console.log(result);
    client.rollback();
    client.disconnect();
```

4.4.8.13 setAutoCommit(Boolean) Method

Changes the autocommit setting for the connection.

Syntax

```
connection.setAutoCommit(flag)
```

Parameters

Type	Name	Description
Boolean	`flag`	Specify a value to enable (true) or disable (false) autocommit. The default is true.

4.4.8.14 setClientInfo(String, String) Method

Sets a client information property for the connection.

Syntax

```
connection.setClientInfo(key, value)
```

Parameters

Type	Name	Description
String	`key`	The property to set.
String	`value`	The value for the property.

Remarks

The supported properties include the session-specific client variables (APPLICATION, APPLICATIONUSER, CLIENT, and USER), user-defined variables, and the properties listed in the M_SESSION_CONTEXT system view.

Related Information

Setting Session-Specific Client Information [page 16]

4.4.8.15 setRowSetSize (Integer) Method

Specifies the size of the row set.

Syntax

```
connection.setRowSetSize(rowSetSize)
```

Parameters

Type	Name	Description
Integer	rowSetSize	Specifies the size of the row set. The value must be 1 (the default) or greater.

Remarks

This method sets the size of the row set that is fetched by Connection.execute or Statement.execute. The default row set size is 1. Increasing the row set size can improve the fetching performance.

> **Note**
>
> Increasing the row set size can result in Connection.execute or Statement.execute returning truncated values when the server returns inaccurate result set metadata.

Related Information

exec[ute](String[, Array][,Object][, Function]) Method [page 544]
exec[ute]([, Array][, Object][, Function]) Method [page 586]

4.4.8.16 state() Method

Returns a string that indicates the state of the connection.

> **Syntax**

```
connection.state( )
```

Returns

Returns *connected* or *disconnected* to indicate the state of the connection.

4.4.9 ConnectionPool Class

Represents an explicit connection pool.

> **Syntax**

```
class ConnectionPool
```

Members

All members of the connection pool.

Methods

Type	Method	Description
	clear([Function]) Method [page 556]	Removes all connections currently in the connection pool.
	getConnection([uid], [pwd], [Function]) Method [page 557]	Retrieves a new connection from the connection pool, creating a new connection if the pool is empty.
	getInUseCount() Method [page 558]	Returns the number of connections created by the connection pool which are currently in use.
	getPooledCount() Method [page 559]	Returns the number of connections currently in the connection pool.
	getProperties() Method [page 560]	Returns the properties of the connection pool.
	setProperties({String | Object}) Method [page 560]	Sets the properties of the connection pool using a connection properties string or a hash of connection properties parameters passed in as a parameter.

Remarks

Used for applications that create connection pools explicitly. Applications that create connection pools implicitly use the `pooling` connection property instead.

> **Note**
>
> Mixed-use applications, applications that use both types of pooling, are not currently supported.

Example

The following SAP HANA Cloud example creates a connection pool:

```
var pool = hana.createPool("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=myusername;pwd=mypassword");
```

Create a connection from the pool, which creates a new connection:

```
var conn = connectionPool.getConnection();
```

Return the connection to the pool, with the connection staying connected to the server:

```
conn.disconnect();
```

Obtain a connection from the pool, which returns the pre-existing connection:

```
conn = connectionPool.getConnection();
```

Return the connection to the pool:

```
conn.disconnect();
```

Empty the pool, causing the connection to be disconnected from the server:

```
pool.clear();
```

Related Information

Node.js Connection Pooling [page 530]
Node.js Connection Properties [page 506]
Explicit Connection Pooling API [page 532]

4.4.9.1 clear([Function]) Method

Removes all connections currently pooled in the connection pool.

Syntax

```
connectionPool.clear([callback] )
```

Parameters

Type	Name	Description
Function	`callback`	Specifies the optional callback function.

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously, as follows:

```
function(err)
{
};
```

Example

The following SAP HANA Cloud example removes all connections in the connection pool:

```
var pool = hana.createPool("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=myusername;pwd=mypassword");
var conn = pool.getConnection();
conn.disconnect();
pool.clear();
```

4.4.9.2 getConnection([uid], [pwd], [Function]) Method

Retrieves a new connection from the connection pool, creating a new connection if the pool is empty.

Syntax

```
connectionPool.getConnection([uid], [pwd], [callback])
```

Parameters

Type	Name	Description
String	`uid`	Specifies the username.
String	`pwd`	Specifies the password.
Function	`callback`	Specifies the optional callback function.

Returns

A connection object is returned.

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously, as follows:

```
function(err, conn)
{
};
```

Example

The following SAP HANA Cloud example creates a connection pool and retrieves a new connection:

```
var pool = hana.createPool("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=myusername;pwd=mypassword");
var conn = pool.getConnection();
```

The following SAP HANA Cloud example creates a connection pool, retrieves a new connection for user1, returns it to the pool, then repurposes it for user2:

```
var pool = hana.createPool("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=user1;pwd=password1");
var conn = pool.getConnection(); // Retrieve a connection for user1.
conn.disconnect(); // Return the connection to the pool.
conn = pool.getConnection("user2", "password2"); // Retrieve a connection for
user2, re-using the connection created earlier, and automatically issue CONNECT
"user2" PASSWORD "password2" to change the user for that connection.
```

4.4.9.3 getInUseCount() Method

Returns the number of connections created by the connection pool which are currently in use.
A connection which is currently is use is defined as a connection which was returned by the `ConnectionPool.getConnection` method and has not called the `Connection.disconnect` method.

Syntax

```
connectionPool.getInUseCount()
```

Parameters

None.

Returns

A number is returned.

Remarks

None.

Example

The following SAP HANA Cloud example returns the number of connections that are currently in use in the connection pool:

```
var pool = hana.createPool("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=myusername;pwd=mypassword");
var numInUseConns = pool.getInUseCount();
```

4.4.9.4 getPooledCount() Method

Returns the number of pooled connections currently in the connection pool. A pooled connection is a connection that is available to be reused by `ConnectionPool.getConnection`.

Syntax

```
connectionPool.getPooledCount()
```

Parameters

None.

Returns

A number is returned.

Remarks

None.

Example

The following SAP HANA Cloud example returns the number of connections in the pool:

```
var pool = hana.createPool("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=myusername;pwd=mypassword");
var numPooledConns = pool.getPooledCount();
```

4.4.9.5 getProperties() Method

Returns the properties of the connection pool.

Syntax

```
connectionPool.getProperties()
```

Parameters

None.

Returns

A string is returned with a semicolon-delimited list of properties.

Remarks

None.

Example

The following SAP HANA Cloud example creates a connection pool and returns its properties:

```
var pool = hana.createPool("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=myusername;pwd=mypassword");
var poolProps = pool.getProperties();
```

4.4.9.6 setProperties({String | Object}) Method

Sets the pool properties of the connection pool using a connection properties string or a hash of connection properties parameters passed in as a parameter. See the createPool({String | Object}[, Object]) Method [page 563] for a description of the available connection pool properties.

Syntax

```
connectionPool.setProperties(pool_parameters)
```

Parameters

Type	Name	Description
String or Object	`<pool_parameters>`	Specifies a valid string of connection pool parameters or a hash of connection property parameters.

Remarks

None.

Example

The following SAP HANA Cloud example creates a connection pool and sets the properties:

```
var pool = hana.createPool("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=myusername;pwd=mypassword");
pool.setProperties("poolCapacity=25;maxConnectedOrPooled=50;maxPooledIdleTime=60;
pingCheck=true");
```

Related Information

createPool({String | Object}[, Object]) Method [page 563]

4.4.10 Hana Class

Provides access to global methods provided by the SAP HANA Node.js driver.

Syntax

```
class Hana
```

Members

All members of the Hana class.

Methods

Type	Method	Description
	{createConnection | createClient}() Method [page 562]	Creates a new connection using a connection string or a hash of connection parameters being passed in as a parameter.
	createPool({String | Object}[, Object]) Method [page 563]	Creates a new connection pool using a connection string or a hash of connection parameters that is passed in as a parameter.
	getDriverVersion() Method [page 566]	Gets the version of the driver.
	setConnectionPoolLimit(Integer) Method [page 566]	Sets the maximum number of connections allowed in the implicit connection pool.

Related Information

Node.js Connection Pooling [page 530]
Explicit Connection Pooling API [page 532]

4.4.10.1 {createConnection | createClient}() Method

Creates a new connection using a connection string or a hash of connection parameters being passed in as a parameter.

Syntax

```
hana.{createConnection | createClient}()
```

Parameters

None.

Returns

A connection object is returned. The `connection.connect` method must be called to establish the connection to the server.

Remarks

None.

Example

The following example creates a connection to the SAP HANA Cloud:

```
var hana=require('@sap/hana-client');
var conn=hana.createConnection();
conn.connect("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=myusername;pwd=mypassword");
```

Related Information

connect({String | Object}[, Function]) Method [page 542]

4.4.10.2 createPool({String | Object}[, Object]) Method

Creates a new explicit connection pool using a connection string or a hash of connection parameters that is passed in as a parameter.

Syntax

```
hana.createPool(<connection_parameters>, [<pool_parameters>])
```

Parameters

Type	Name	Description
String or object	`<connection_parameters>`	Specifies a valid connection string or a hash of connection parameters.
String or object	`<pool_parameters>`	Specifies an optional set of explicit connection pool properties.

Connection Pool Properties

Property	Value	Default	Description
`allowSwitchUser`	Boolean	FALSE	Determines whether or not the pooled connection will allow the pool to contain connections from more than one user.
`maxConnectedOrPooled`	`<number>`	0	Defines the maximum number of open connections created by the pool, each of which may be in the pool waiting to be reused or may no longer be in the pool and actively being used. Specifying 0 (the default) indicates that there is no limit to the number of connections. **Note** There are two states of connections obtained from a connection pool, pooled connections and in-use connections. Pooled connections are connections which are in a pool and prepared for reuse. In-use connections are connections that are currently being used and which are returned to the pool once disconnected.
`maxPooledIdleTime`	`<number>`	0	Defines the maximum time, in seconds, that connections are allowed to remain in the pool before being marked for eviction. Specifying 0 (the default) indicates that there is no limit to the time a connection can remain in the pooled state.

Property	Value	Default	Description
`pingCheck`	Boolean	FALSE	Determines whether or not the pooled connection should be tested for viability before being reused.
`poolCapacity`	`<number>`	0	Defines the maximum number of connections allowed to be in the pool, waiting to be reused. Specifying 0 (the default) indicates that there is no limit to the number of connections.

Returns

An explicit `connectionPool` object is returned.

Remarks

By default, the Node.js driver does not have explicit connection pooling enabled. Explicit connection pooling can be enabled either by using this explicit API or by setting the `<pooling>` connection property to TRUE when creating a connection. These two approaches are mutually exclusive.

Example

The following example creates a new explicit connection pool to the SAP HANA Cloud:

```
var hana = require('@sap/hana-client');
var pool = hana.createPool("ServerNode=myhost-
hanacloud.ondemand.com:443;uid=myusername;pwd=mypassword",
"poolCapacity=25;maxConnectedOrPooled=50;maxPooledIdleTime=60;pingCheck=true");
var conn = pool.getConnection();
var results = conn.execute("SELECT * FROM M_DATABASE");
conn.disconnect();
```

Related Information

Node.js Connection Pooling [page 530]

Node.js Connection Properties [page 506]

4.4.10.3 getDriverVersion() Method

Gets the version of the driver.

Syntax

```
hana.getDriverVersion()
```

Parameters

None.

Returns

The driver version is returned as a string, for example Node.js driver version: 2.12.025.1651858745.

Remarks

None.

Example

The following example gets the driver version:

```
var driverVersionStr = hana.getDriverVersion();
```

4.4.10.4 setConnectionPoolLimit(Integer) Method

Sets the maximum number of connections allowed in the implicit connection pool.

Syntax

```
hana.setConnectionPoolLimit(limit)
```

Parameters

Type	Name	Description
Integer	`limit`	Specifies a positive number to set the limit to. This number cannot be negative.

Remarks

If an active pooled connection, meaning a connection that has the `pooling`=true connection property, is disconnected while the implicit connection pool has reached the specified limit, the connection is closed, as opposed to being added to the connection pool.

Example

The following example sets the maximum number of connections in the connection pool to 100:

```
hana.setConnectionPoolLimit(100);
```

4.4.11 ResultSet Class

Represents a result set.

Syntax

```
class ResultSet
```

Members

All members of ResultSet.

Methods

Method	Description
close() [page 569]	Closes the ResultSet object and frees up resources.
getColumnCount() [page 569]	Gets the number of columns in the current result set.

Method	Description
getColumnInfo() Method [page 570]	Gets specified column information.
getColumnName(Integer) [page 570]	Gets the name of the column for the specified zero-based column index.
getData(Integer, Integer, Buffer, Integer, Integer[, Function]) [page 571]	Reads a stream of bytes from the specified LOB column, starting at the location indicated by dataOffset, into the buffer, starting at the location indicated by bufferOffset.
getRowCount() Method [page 572]	Gets the number of rows in the current result set.
getServerCPUTime() Method [page 573]	Returns the server CPU time for the result set fetch.
getServerMemoryUsage() Method [page 574]	Returns the server memory usage for the result set fetch.
getServerProcessingTime() Method [page 575]	Returns the elapsed server processing time for the result set fetch.
getValue(Integer) [page 576]	Gets the value of the specified column.
getValuesCanBlock() Method [page 577]	Checks if calling the `getValues` method on the current row will block.
getValueLength(Integer) Method [page 578]	Gets the length of a LOB column or a character string type.
getValues() [page 579]	Gets an object with the column names and values of the current row.
isClosed() Method [page 581]	Checks if the result set is closed.
isNull(Integer) Method [page 581]	Checks if the specified column is NULL.
next([Function]) [page 582]	Gets the next row in the result set.
nextResult([Function]) [page 583]	Gets the next result set.

Remarks

> **Note**
>
> If there is only one result set, and you are fetching the entire result set, then `Statement.exec[ute]` has much better performance than `Statement.exec[ute]Query` followed by using the `ResultSet` object to process the result set. However, for result sets containing more than a few hundred kilobytes of data, `Statement.exec[ute]` can use significantly more memory.

Related Information

exec[ute]Query([Array][, Object][, Function]) Method [page 590]
exec[ute]([, Array][, Object][, Function]) Method [page 586]

4.4.11.1 close([Callback]) Method

Closes the ResultSet object and frees up resources.

Syntax

```
resultset.close([callback])
```

Parameters

Type	Name	Description
Function	`callback`	Specifies the optional callback function.

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously.

The callback function is of the following form:

```
function(err){
};
```

4.4.11.2 getColumnCount() Method

Gets the number of columns in the current result set.

Syntax

```
resultset.getColumnCount()
```

Returns

Returns the number of columns in the current result set.

4.4.11.3 getColumnInfo() Method

Gets information about each column in the result set.

Syntax

```
ResultSet.getColumnInfo()
```

Returns

Returns an array of objects. There is an object that contains the following properties for each column in the result set: `<columnName>`, `<tableName>`, `<ownerName>`, `<type>`, `<typeName>`, `<nativeType>`, `<nativeTypeName>`, `<precision>`, `<scale>`, and `<nullable>`.

4.4.11.4 getColumnName(Integer) Method

Gets the name of the column for the specified zero-based column index.

Syntax

```
resultset.getColumnName(colIndex)
```

Parameters

Type	Name	Description
Integer	`colIndex`	The zero-based column index.

Returns

Returns the name of the column.

4.4.11.5 getData(Integer, Integer, Buffer, Integer, Integer[, Function]) Method

Reads a stream of bytes from the specified LOB column, starting at the location indicated by dataOffset, into the buffer, starting at the location indicated by bufferOffset.

Syntax

```
resultset.getData(colIndex, dataOffset, buffer, bufferOffset, length[,
callback])
```

Parameters

Type	Name	Description
Integer	`colIndex`	The zero-based column index.
Integer	`dataOffset`	The index within the column where the read operation begins. For binary columns this value is in bytes. For string columns this value is in UTF-16 encoded characters.
Buffer	`buffer`	The buffer into which to copy the data.
Integer	`bufferOffset`	The index in bytes within the buffer to which the data is copied.
Integer	`length`	The maximum number of bytes to read.
Function	`callback`	The optional callback function.

Returns

Returns the actual number of bytes read.

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously.

The callback function is of the following form:

```
function(err, result){
};
```

For string columns, the data filled into the buffer is UTF-8 encoded. The driver will not copy partial UTF-8 characters. Because of this, the number of bytes returned for a string column can be a few bytes less than the length parameter, even when there is more data available in the column.

Note that the `dataOffset` for the column is in UTF-16 units for strings, so non-BMP characters count as two UTF-16 units. The example below shows that Node.js string lengths are in UTF-16 units and can be used to compute the `dataOffset` parameter when getting multiple adjacent parts of column data:

Sample Code

```
var hana=require('@sap/hana-client');
var client=hana.createConnection();
client.connect("serverNode=myServer:30015;uid=system;pwd=manager");
// NCLOB_COL is of type NCLOB
var stmt=client.prepare("SELECT TOP 1 NCLOB_COL FROM LOBTEST ORDER BY PK");
var rs=stmt.execQuery();
if(rs.next()) {
    var buffer_size = 1000000;
    var buffer = Buffer.alloc(buffer_size);
    var offset = 0; // offset in UTF-16 character units into column
    var total_bytes_read = 0;
    while(true) {
        var bytes_read = rs.getData(0, offset, buffer, 0, buffer_size);
        if(bytes_read == 0) break;
        var got_str = buffer.toString('utf8', 0, bytes_read);
        console.log("getData got: " + bytes_read + " bytes");
        console.log("  (" + got_str.length + " characters)"); // in UTF-16
units
        console.log("  starting at: " + offset);
        console.log("  first character retreived is: " + got_str[0]);
        console.log("  last character retreived is:" +
got_str[got_str.length-1]);
        total_bytes_read += bytes_read;
        offset += got_str.length;
    }
    console.log("Done getting data");
    console.log("  total characters read is " + offset);
    console.log("  total bytes read is " + total_bytes_read);
}
rs.close();
stmt.drop();
client.disconnect();
```

4.4.11.6 getRowCount() Method

Gets the number of rows in the current result set.

Syntax

```
resultset.getRowCount()
```

Returns

This function returns the number of rows in the current result set. In many cases, the client does not know the total number of rows. Returns -1 if unknown, 0 if there are no rows, or <n>, with <n> specifying the exact number of rows valued at greater than 0. A positive <n> is only available after the app has fetched the last chunk of rows.

4.4.11.7 getServerCPUTime() Method

Returns the server CPU time for the result set fetches.

Syntax

```
ResultSet.getServerCPUTime()
```

Returns

The server CPU time for the result set fetches, in microseconds. This value does not include network communication time.

Remarks

The returned value is for the fetches since the beginning of the current result set. If nextResult is called, then the returned result is reset for the new result set.

By default, executed statements include server CPU time for the first 32 rows. If fetching more rows requires additional fetch requests to the server, then the server CPU time value is a non-zero value and calculated from the beginning of the current result set.

Example

Connect to the SAP HANA database:

```
var hana=require('@sap/hana-client');
    var conn=hana.createConnection();
    conn.connect("serverNode=myserver;uid=<user-id>;pwd=<password>");
```

Run the following statement with the `getServerCPUTime()` method to retrieve the server CPU time for the fetch requests (not including the execute server CPU time):

```
var sql = "SELECT TOP 100 * FROM OBJECTS";
```

```
var stmt = conn.prepare(sql);
var result_set = stmt.execQuery();
var rows = [];
while (result_set.next()) {
    rows.push(result_set.getValues());
}
var cpuTime = result_set.getServerCPUTime();
```

4.4.11.8 getServerMemoryUsage() Method

Returns the server memory usage for the result set fetches.

Syntax

```
ResultSet.getServerMemoryUsage()
```

Returns

The server memory usage for the result set fetches, in bytes.

Remarks

The returned value is for the fetches since the beginning of the current result set. If nextResult is called, then the returned results are reset for the new result set.

By default, executed statements include server memory usage for the first 32 rows. If fetching more rows requires additional fetch requests to the server, then the server memory usage value is a non-zero value and calculated from the beginning of the current result set.

`getServerMemoryUsage()` returns 0 if you are connected to an SAP HANA 2.0 SPS 03 server or earlier. It also returns 0 unless memory_tracking and resource_tracking is enabled in the resource_tracking section of global.ini by setting enable_tracking and memory_tracking to "on". For performance reasons, resource tracking is disabled by default.

Example

Ensure that monitoring is enabled in the resource_tracking section of `global.ini` by setting enable_tracking and memory_tracking to "on". For performance reasons, resource tracking is disabled by default.

Connect to the SAP HANA database:

```
var hana=require('@sap/hana-client');
var conn=hana.createConnection();
```

```
conn.connect("serverNode=myserver;uid=<user-id>;pwd=<password>");
```

Run the following statement with the `getServerMemoryUsage` method to retrieve the server memory usage for the fetch requests (not including the execute server memory usage):

```
var sql = "SELECT TOP 100 * FROM OBJECTS";
var stmt = conn.prepare(sql);
var result_set = stmt.execQuery();
var rows = [];
while (result_set.next()) {
    rows.push(result_set.getValues());
}
var memoryUsage = result_set.getServerMemoryUsage();
```

4.4.11.9 getServerProcessingTime() Method

Returns the elapsed server processing time for the result set fetches.

Syntax

```
ResultSet.getServerProcessingTime()
```

Returns

The elapsed server processing time for the result set fetches, in microseconds. This value does not include network communication time.

Remarks

The returned value is for the fetches since the beginning of the current result set. If nextResult is called, then the returned result is reset for the new result set.

By default, executed statements include fetch server processing time for the first 32 rows. If fetching more rows requires additional fetch requests to the server, then the server processing time is a non-zero value and calculated from the beginning of the current result set.

Example

Connect to the SAP HANA database:

```
var hana=require('@sap/hana-client');
var conn=hana.createConnection();
```

```
conn.connect("serverNode=myserver;uid=<user-id>;pwd=<password>");
```

Run the following statement with the `getServerProcessingTime()` method to retrieve the server processing time for the fetch requests (not including the execute server processing time):

```
var sql = "SELECT TOP 100 * FROM OBJECTS";
var stmt = conn.prepare(sql);
var result_set = stmt.execQuery();
var processingTime = result_set.getServerProcessingTime();
var rows = [];
while (result_set.next()) {
    rows.push(result_set.getValues());
}
var processingTime = result_set.getServerProcessingTime();
```

4.4.11.10 getValue(Integer) Method

Gets the value of the specified column.

Syntax

```
resultset.getValue(colIndex)
```

Parameters

Type	Name	Description
Integer	`colIndex`	The zero-based column index.

Returns

Returns the value of the column.

Remarks

`ResultSet.getValues` has significantly better performance than calling `ResultSet.getValue` for all columns.

Example

The following example shows how to use the `getValue` method synchronously:

```
var hana = require('@sap/hana-client')
var conn = hana.createConnection();
conn.connect("serverNode=myserver:30015;uid=system;pwd=manager");
var stmt = conn.prepare("select * from M_SERVICES");
var resultset = stmt.executeQuery();
var numcolumns = resultset.getColumnCount();
while(resultset.next()) {
    for(var i = 0; i < numcolumns; i++) {
        console.log("column %d: ", i+1, resultset.getValue(i));
    }
}
resultset.close();
```

Related Information

getValues([Options][, Callback]) Method [page 579]

4.4.11.11 getValuesCanBlock() Method

Checks if calling the `getValues` method on the current row will block.

Syntax

```
resultset.getValuesCanBlock()
```

Returns

This method returns true if calling `getValues` on the current row will block. The method returns false if calling `getValues` will not block.

When true is returned, `getValues` should be called asynchronously. For significantly better performance, when false is returned, it is safe and recommended to call `getValues` synchronously.

Remarks

The `getValuesCanBlock()` method returns true if the row contains a LOB value that is more than approximately 1KB. Note that in earlier versions of the @sap/hana_client driver, the `getValues` method could block in many more cases.

When `getValuesCanBlock()` returns false, there are not particularly large LOB values in the row (never more than 10K or so). You will not use a lot of memory by calling `getValues()`.

Example

The following example shows that you can use `getValuesCanBlock()` to determine if `getValues` should be called synchronously or asynchronously in order to maximize performance. This example uses the `Promise.js` extension.

```
'use strict';
var hana = require('@sap/hana-client')
var PromiseModule = require('@sap/hana-client/extension/Promise.js');
var conn = hana.createConnection();
async function promiseFetchAll(conn) {
    await PromiseModule.connect(conn,
"ServerNode=myserver:30115;uid=system;pwd=manager");
    var stmt = await PromiseModule.prepare(conn, "select * from M_SERVICES");
    var resultset = await PromiseModule.executeQuery(stmt);
    while(await PromiseModule.next(resultset)) {
        var row;
        var canBlock = resultset.getValuesCanBlock();
        if(canBlock) {
            // asynchronous
            row = await PromiseModule.getValues(resultset);
        } else {
            // synchronous
            row = resultset.getValues();
        }
        console.log(canBlock.toString(), row);
    }
    await PromiseModule.close(resultset);
    await PromiseModule.drop(stmt);
    await PromiseModule.disconnect(conn);
}
promiseFetchAll(conn);
```

Related Information

4.4.11.12 getValueLength(Integer) Method

Gets the length of a LOB column or a character string type.

Syntax

```
resultset.getValueLength(colIndex)
```

Parameters

Type	Name	Description
Integer	`colIndex`	The zero-based column index.

Returns

Returns the byte length of a LOB or character string type, or -1 if that information is not available. For NCLOB values, this length is in UTF-16 units (so a non-BMP characters counts as two characters). For NVARCHAR values, this length is the byte length in the UTF-8 encoding (which is different from the JavaScript string length). If the column is a NULL value, then 0 is returned.

4.4.11.13 getValues([Options][, Callback]) Method

Gets an object with the column names and values of the current row.

Syntax

```
resultset.getValues([options] [, callback])
```

Parameters

Type	Name	Description
Object	`options`	Specifies the optional object containing the options for defining the representation of rows in the result.
Function	`callback`	Specifies the optional callback function.

Returns

If the options parameter is specified as `{asArray: true}`, this method returns an array consisting of all column values for the current row. Otherwise, it returns an object that consists of a set of column names and value pairs for the current row. If multiple columns have the same column name, the property name for the second and subsequent columns with the same column name are "`<column-name>:<n>`" where `<n>` is a number starting from 1.

If no callback is specified, then the result is returned.

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously.

The callback function is of the following form:

```
function(err, result){
};
```

Example

The following example shows how to use the `getValues` method synchronously:

```
var hana = require('@sap/hana-client')
var conn = hana.createConnection();
conn.connect("serverNode=myserver:30015;uid=system;pwd=manager");
var stmt = conn.prepare("select * from M_SERVICES");
var resultset = stmt.executeQuery();
while(resultset.next()) {
    var row = resultset.getValues({asArray: true});
    for(var i = 0; i < row.length; i++) {
        console.log("column %d: ", i+1, row[i]);
    }
    console.log();
}
resultset.close();
```

The following example shows how to use the `getValues` method asynchronously:

```
var hana = require('@sap/hana-client')
var conn = hana.createConnection();
conn.connect("serverNode=myserver:30015;uid=system;pwd=manager");
var stmt = conn.prepare("select * from M_SERVICES");
var resultset = stmt.executeQuery();
processrows(resultset);
function processrows(resultset) {
    resultset.next(function(err, result) {
        if(err) {
            console.log("next error", err);
        } else if(result) {
            resultset.getValues({asArray: true}, function(err, row) {
                if(err) {
                    console.log("getValues error", err);
                } else {
                    for(var i = 0; i < row.length; i++) {
                        console.log("column %d: ", i+1, row[i]);
                    }
                    console.log();
                    processrows(resultset);
                }
            });
        } else {
            resultset.close(function(err) {
```

```
                // done processing the result set
            });
        }
    });
}
```

4.4.11.14 isClosed() Method

Checks if the result set is closed.

Syntax

```
resultset.isClosed()
```

Returns

This function returns true if the result set is closed and false otherwise.

4.4.11.15 isNull(Integer) Method

Checks if the specified column is NULL.

Syntax

```
resultset.isNull(colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	Specifies the zero-based column index.

Returns

A Boolean that indicates whether the specified column is NULL.

4.4.11.16 next([Function]) Method

Gets the next row in the result set.

Syntax

```
resultset.next([callback])
```

Parameters

Type	Name	Description
Function	`callback`	Specifies the optional callback function.

Returns

Returns true if there are more rows in the result set and false otherwise.

Remarks

This method attempts to fetch the next row of the result set, returning `true` when successful and `false` if there are no more rows.

When a result set is initially created by `Statement.exec[ute]Query`, the result set is positioned before the first row and must next be called before getting the first row values. Similarly, after calling `nextResult`, the result set is positioned before the first row.

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously.

The callback function takes the following form:

```
function(err, result){
};
```

4.4.11.17 nextResult([Function]) Method

Gets the next result set.

Syntax

```
resultset.nextResult([callback])
```

Parameters

Type	Name	Description
Function	`callback`	Specifies the optional callback function.

Returns

Returns true if there are more result sets and false otherwise.

Remarks

This method checks to see if there are more result sets and fetches the next available result set. This method supports asynchronous callbacks.

4.4.12 Statement Class

Represents a prepared statement.

Syntax

```
class Statement
```

Members

All members of Statement.

Methods

Type	Method	Description
	drop([Function]) [page 585]	Drops the prepared SQL statement.
	exec[ute]([, Array][, Object][, Function]) Method [page 586]	Executes the prepared SQL statement.
	exec[ute]Batch(Array[, Function]) [page 588]	Submits the command for batch execution and if the command executes successfully, then this method returns the number of rows affected.
	exec[ute]Query([Array][, Function]) [page 590]	Executes the prepared SQL statement and returns a result set object.
	functionCode() [page 593]	Gets the function code of the statement.
	getColumnInfo() Method [page 591]	Gets specified column information.
	getData(Integer, Integer, Buffer, Integer, Integer[, Function]) Method [page 592]	Reads a stream of bytes from the specified output LOB parameter.
	getPacketSize() Method	Retrieves the network packet size for the statement.
	getParameterInfo() Method [page 593]	Gets the specified parameter information.
	getParameterValue(Integer) [page 591]	Retrieves output parameter values.
	getPrintLines() Method [page 594]	Allows applications to access to messages printed from a stored procedure via SQLSCRIPT_PRINT:PRINT_LINE.
	getRowStatus() Method [page 595]	Returns the status array of the most recent batch execution.
	getServerCPUTime() Method [page 595]	Retrieves the total CPU processing time.
	getServerMemoryUsage() Method [page 596]	Returns memory usage data for a prepared or executed statement.
	getServerProcessingTime() Method [page 597]	Returns the elapsed processing time for a prepared or executed statement.
	isValid() Method [page 598]	Checks whether the statement is valid.
	sendParameterData(columnIndex, buffer[, callback]) Method [page 598]	Sends LOB data to the server in chunks.
	setPacketSize(Integer) Method	Sets the network packet size for the statement.
	setTimeout(Integer) Method [page 600]	Changes the default timeout for a statement.

Remarks

The Statement object is for SQL statements that are executed multiple times.

Related Information

prepare(String[, Function]) Method [page 550]
Packet Size [page 821]

4.4.12.1 drop([Function]) Method

Drops the prepared SQL statement.

Syntax

```
statement.drop ([callback])
```

Parameters

Type	Name	Description
Function	`callback`	The optional callback function.

Remarks

This method drops the prepared statement and frees up resources.

This method can be either synchronous or asynchronous depending on whether or not a callback function is specified. The callback function is of the form:

```
function(err)
    {
    };
```

Example

The following synchronous example shows how to use the drop method on a prepared statement.

```
var hana=require('@sap/hana-client');
    var client=hana.createConnection();
    client.connect("serverNode=myserver;uid=system;pwd=manager");
    stmt=client.prepare("SELECT * FROM Customers WHERE ID >= ? AND ID < ?");
    result=stmt.exec([200, 300]);
    stmt.drop();
    console.log(result);
    client.disconnect();
```

4.4.12.2 exec[ute]([, Array][, Object][, Function]) Method

Executes the prepared SQL statement.

Syntax

```
statement.exec[ute] ([params][, options][, callback])
```

Parameters

Type	Name	Description
Array	`parameters`	Specifies the optional array of bind parameters.
Object	`options`	Specifies the optional object containing the options.
Function	`callback`	Specifies the optional callback function.

Returns

For queries that produce result sets, this method returns an array of row objects. The **Options** field defines the representation of each row, along with options applicable to the query as a whole. The following options can be used:

- The default row representation is an object where each property name is the column name and the property value is the column value. If multiple columns have the same column name, the property name for the second and subsequent columns with the same column name is "`<column-name>:<n>`" where `<n>` is a number starting from 1.
- {rowsAsArray: true} makes the row representation an array of column values. Using this format may result in improved performance. Option keys can be nestTables and rowsAsArray.

- {nestTables: true} makes the row representation a nested object. If the table names for columns are known, the object property names are the table names. The object property values are themselves objects, where each property name is the column name, and the property value is the column value. If the table names for columns are not known, then the representation is the same as the default representation.
- {communicationTimeout: int} ensures that the client will time out within the specified time, in milliseconds, when a connection is queried.

If no callback is specified, then the result is returned.

> **Note**
>
> For queries, the entire result set is returned as an array of rows. Since large result sets can use significant memory and time, it is recommended that a WHERE or LIMIT clause is used where possible. The `rowsAsArray` option is recommended for best performance.

For INSERT, UPDATE, and DELETE statements, the number of rows affected is returned as the second parameter of the callback function.

For other statements, the result is undefined.

> **Note**
>
> This method returns the first result set. To return more than one result set use exec[ute]Query([Array][, Object][, Function]) Method [page 590]. To move from one result set to the next use nextResult([Function]) Method [page 583].

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err, ret){
};
```

Example

The following synchronous example shows how to use the exec method on a prepared statement.

```
var hana=require('@sap/hana-client');
var client=hana.createConnection();
client.connect("serverNode=myserver;uid=system;pwd=manager");
stmt=client.prepare("SELECT * FROM Customers WHERE ID >= ? AND ID < ?");
result=stmt.exec([200, 300]);
stmt.drop();
console.log(result);
client.disconnect();
```

To return a separate object per row for each table when you specify a SQL statement as a join with overlapping column names, set the `nestTables` option to true. For example:

```
var options = {nestTables: true};
stmt.exec(options, function(err, rows)
{   // process rows
});
```

Specifying the above example returns an array like the following example:

```
[{
  T1: {
   ID: 1,
   A: 't1.1.a',
   B: 't1.1.b'
  },
  T2: {
   ID: 1
   A: 't2.1.a',
   B: 't2.1.b'
  }
}]
```

To return all rows as an array where the order of the column values is exactly the same as in Statement.getColumnInfo, set the `rowsAsArray` option to true. For example:

```
stmt.exec(command, {rowsAsArray: true}, function(err, rows) { /* process rows
*/ });
```

Specifying the above example returns an array like the following example:

```
[[
  1,
  't1.1.a',
  't1.1.b',
  1,
  't2.1.a',
  't2.1.b'
]]
```

4.4.12.3 exec[ute]Batch([Array][, Object][, Function]) Method

Submits the command for batch execution and if the command executes successfully, then this method returns the number of rows affected.

Syntax

```
statement.exec[ute]Batch ([params][, callback])
```

Parameters

Type	Name	Description
Array	`parameters`	Specifies an array of bind parameters to execute.
Object	`options`	Specifies the optional object containing the options.
Function	`callback`	Specifies the optional callback function.

Returns

If you do not specify a callback function, then the number of rows affected is returned.

For queries that produce result sets, this method returns an array of row objects. The {communicationTimeout: int} option ensures that the client will time out within the specified time, in milliseconds, when a connection is queried.

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err, ret){
   };
```

Example

The following synchronous example shows how to use the exec method on a prepared statement.

```
var hana=require('@sap/hana-client');
   var client=hana.createConnection();
   client.connect("serverNode=myserver;uid=system;pwd=manager");
   stmt=client.prepare("INSERT INTO Customers(ID, NAME) VALUES(?, ?)");
   result=stmt.execBatch([[1, 'Company 1'], [2, 'Company 2']]);
   stmt.drop();
   console.log(result);
   client.disconnect();
```

4.4.12.4 exec[ute]Query([Array][, Object][, Function]) Method

Executes the prepared SQL statement and returns a result set object.

Syntax

```
statement.exec[ute]Query ([parameters][, callback])
```

Parameters

Type	Name	Description
Array	`parameters`	Specifies an optional array of bind parameters to execute.
Object	`options`	Specifies the optional object containing the options.
Function	`callback`	Specifies the optional callback function.

Returns

The result set object is returned as the second parameter of the callback. If no callback is specified, then the result set is returned.

For queries that produce result sets, this method returns an array of row objects. The {communicationTimeout: int} option ensures that the client will time out within the specified time, in milliseconds, when a connection is queried.

Remarks

This method can be synchronous or asynchronous. Specify a callback function to use this method asynchronously. The callback function is of the form:

```
function(err, resultset){
};
```

Note

If there is only one result set, and you are fetching the entire result set, then `Statement.exec[ute]` has much better performance than `Statement.exec[ute]Query` followed by using the `ResultSet` object to process the result set. However, for result sets containing more than a few hundred kilobytes of data, `Statement.exec[ute]` can use significantly more memory.

Example

The following synchronous example shows how to use the execQuery method on a prepared statement.

```
var hana=require('@sap/hana-client');
var client=hana.createConnection();
client.connect("serverNode=myserver;uid=system;pwd=manager");
stmt=client.prepare("SELECT * FROM Customers WHERE ID >= ? AND ID < ?");
result=stmt.execQuery([200, 300]);
stmt.drop();
console.log(result);
client.disconnect();
```

Related Information

ResultSet Class [page 567]

4.4.12.5 functionCode() Method

Gets the function code of the statement.

Syntax

```
statement.functionCode()
```

Returns

Returns the statement's function code.

4.4.12.6 getColumnInfo() Method

Returns the metadata of all of the columns in the result set when the statement is executed.

Syntax

```
statement.getColumnInfo()
```

Returns

Returns an array of objects. There is an object that contains the following properties for each column: `<columnName>`, `<tableName>`, `<ownerName>`, `<type>`, `<typeName>`, `<nativeType>`, `<nativeTypeName>`, `<precision>`, `<scale>`, and `<nullable>`. If the result set is empty, then an empty array is returned.

4.4.12.7 getData(Integer, Integer, Buffer, Integer, Integer[, Function]) Method

Reads a stream of bytes from the specified output LOB parameter.

Syntax

```
statement.getData(paramIndex, dataOffset, buffer, bufferOffset, length[,
callback])
```

Parameters

Type	Name	Description
Integer	`paramIndex`	Specifies the zero-based parameter index.
Integer	`dataOffset`	Specifies the index within the parameter where the read operation begins.
Buffer	`buffer`	Specifies the buffer into which to copy the data.
Integer	`bufferOffset`	Specifies the index within the buffer to which the data is copied.
Integer	`length`	Specifies the maximum number of bytes to read.
Function	`callback`	Specifies the optional callback function.

Returns

Returns the actual number of bytes read.

Remarks

This method supports asynchronous callbacks.

4.4.12.8 getParameterInfo() Method

Gets information about each bind parameter used by the statement.

Syntax

```
statement.getParameterInfo()
```

Returns

Returns an array of objects. There is an object that contains the following properties for each parameter in the result set: `<name>`, `<direction>`, `<nativeType>`, `<nativeTypeName>`, `<precision>`, `<scale>`, `<maxSize>`, `<type>`, `<typeName>`. If there are no parameters an empty array is returned.

4.4.12.9 getParameterValue(Integer[, Function]) Method

Retrieves output parameter values.

Syntax

```
statement.getParameterValue (paramIndex [, callback])
```

Parameters

Type	Name	Description
Integer	`paramIndex`	Specifies the zero-based index of the parameter.
Function	`callback`	Specifies the optional callback function.

Returns

Returns the value of the specified parameter.

4.4.12.10 getPrintLines() Method

Allows applications access to messages printed from a stored procedure via SQLSCRIPT_PRINT:PRINT_LINE.

Syntax

```
statement.getPrintLines( )
```

Returns

An array that contains all print lines that were printed during the execution of a stored procedure.

Remarks

This method makes messages printed from a stored procedure via SQLSCRIPT_PRINT:PRINT_LINE accessible to applications.

Example

The following example returns strings from the TEST_PROC procedure:

```
CREATE PROCEDURE TEST_PROC AS
BEGIN
    USING SQLSCRIPT_PRINT AS PRINT;
    SELECT 1 FROM DUMMY;
    PRINT:PRINT_LINE('Hello, World!');
    SELECT 2 FROM DUMMY;
    PRINT:PRINT_LINE('Hello World, Again!');
END
var stmt = conn.prepare('CALL TEST_PROC');
var rs = stmt.execQuery();
var pls = stmt.getPrintLines();
console.log(pls);
```

4.4.12.11 getRowStatus() Method

Returns the status array of the most recent batch execution.

Syntax

```
statement.getRowStatus()
```

Returns

Returns an array of integers. The integers describe the state of each row: 1 means succeeded and -2 means failed. Returns undefined if the last execute was not an exec[ute]Batch with more than one row of parameters.

4.4.12.12 getServerCPUTime() Method

Retrieves the total server CPU processing time.

Syntax

```
Statement.getServerCPUTime()
```

Returns

The server CPU processing time that it took to prepare or execute statements, in microseconds. This does not include network communication time.

Remarks

If a statement has been prepared but not executed, then the returned value is only for the prepare. If the statement has been executed, then the returned value is only for the most recent execution. By default, the execution server CPU time includes the fetch server CPU time for the first 32 rows.

If connected to a HANA 2 SPS 03 or earlier server, `Statement.getServerCPUTime()` always returns 0.

Example

Connect to the SAP HANA database:

```
var hana=require('@sap/hana-client');
    var conn=hana.createConnection();
    conn.connect("serverNode=myserver;uid=<user-id>;pwd=<password>");
```

Run the following statement with the `getServerCPUTime` method to retrieve the server CPU processing time for the prepare and execute phases of the statement execution:

```
var sql = "SELECT TOP 100 * FROM OBJECTS";
var stmt = conn.prepare(sql);
var cpuTimePrepare = stmt.getServerCPUTime();
var result_set = stmt.execQuery();
var cpuTimeExec = stmt.getServerCPUTime();
```

4.4.12.13 getServerMemoryUsage() Method

Returns server memory usage for a prepared or executed statement.

Syntax

```
Statement.getServerMemoryUsage()
```

Returns

Memory usage for prepared or executed statements on the server side, in bytes.

Remarks

If a statement has been prepared but not executed, then the returned value is only for the prepare. If the statement has been executed, then the returned value is only for the most recent execution. By default, executed statements include fetchserver memory usage for the first 32 rows.

`getServerMemoryUsage`() returns 0 if connected to a HANA 2 SPS 03 or earlier server and also returns 0 unless memory_tracking and resource_tracking is enabled in the resource_tracking section of global.ini by setting enable_tracking and memory_tracking to "on". For performance reasons, resource tracking is disabled by default.

Example

Ensure that monitoring is enabled in the resource_tracking section of `global.ini` by setting enable_tracking and memory_tracking to "on". For performance reasons, resource tracking is disabled by default.

Connect to the SAP HANA database:

```
var hana=require('@sap/hana-client');
    var conn=hana.createConnection();
    conn.connect("serverNode=myserver;uid=<user-id>;pwd=<password>");
```

Run the following statement with the `getServerMemoryUsage` method to retrieve the server memory usage for the prepare and execute phases of statement execution:

```
var sql = "SELECT TOP 100 * FROM OBJECTS";
var stmt = conn.prepare(sql);
var memoryUsagePrepare = stmt.getServerMemoryUsage();
var result_set = stmt.execQuery();
var memoryUsageExec = stmt.getServerMemoryUsage();
```

4.4.12.14 getServerProcessingTime() Method

Returns the elapsed server processing time for a prepared or executed statement.

Syntax

```
Statement.getServerProcessingTime()
```

Returns

The elapsed processing time for a prepared or executed statement on the server side, in microseconds. This value does not include network communication time.

Remarks

If a statement has been prepared but not executed, then the returned value is only for the prepared statement. If the statement has been executed, then the returned value is only for the most recent execution.

By default, executed statements include the server fetch processing time for the first 32 rows.

Example

Connect to the SAP HANA database:

```
var hana=require('@sap/hana-client');
var conn=hana.createConnection();
conn.connect("serverNode=myserver;uid=<user-id>;pwd=<password>");
```

Run the following statement to retrieve the server processing time for the prepare and execute operations (including the server time to fetch the first 32 rows by default):

```
var sql = "SELECT TOP 100 * FROM OBJECTS";
var stmt = conn.prepare(sql);
var processingTimePrepare = stmt.getServerProcessingTime();
var result_set = stmt.execQuery();
var processingTimeExec = stmt.getServerProcessingTime();
```

4.4.12.15 isValid() Method

Checks whether the statement is valid.

Syntax

```
statement.isValid()
```

Returns

Returns true if valid, and false otherwise (the statement has been dropped or is being dropped).

4.4.12.16 sendParameterData(paramIndex, buffer[, callback]) Method

Sends LOB data to the server in chunks.

Syntax

```
Statement.sendParameterData(paramIndex, buffer[, callback])
```

Parameters

Type	Name	Description
Integer	`paramIndex`	Specifies the index of the LOB parameter.
Buffer	`buffer`	Specifies the data in the buffer is sent to the server.
Function	`callback`	Specifies the optional callback function.

Returns

Returns a Boolean value that indicates whether the method succeeded.

Remarks

Before calling sendParameterData, call the execute method to specify the sendParameterData to true for each LOB parameter.

Example

The following example shows how to call the execute method before using sendParameterData:

```
var conn = openConnection();
var buffer = new Buffer(5000);
for (var i = 0; i < buffer.length; i++) {
    buffer[i] = 97;
}
var id = 1;
conn.setAutoCommit(false);
var stmt = conn.prepare('INSERT INTO TEST_TABLE(ID, C_BLOB, C_CLOB)
VALUES(?, ?, ?)');
stmt.exec([id, { sendParameterData: true }, { sendParameterData: true }]);
stmt.sendParameterData(1, buffer);
stmt.sendParameterData(1, buffer);
stmt.sendParameterData(1, null);
stmt.sendParameterData(2, buffer);
stmt.sendParameterData(2, buffer);
stmt.sendParameterData(2, buffer);
stmt.sendParameterData(2, null);
conn.commit();
```

Related Information

exec[ute]([, Array][, Object][, Function]) Method [page 586]

4.4.12.17 setTimeout(Integer) Method

Changes the default timeout for a statement.

Syntax

```
statement.setTimeout (timeout)
```

Parameters

Type	Name	Description
Integer	`timeout`	The timeout value in seconds. The default value is 0.

4.4.13 Stream Module

A node.js module that exposes streaming functions.

Syntax

```
var stream = require('@sap/hana-client/extension/Stream.js');
```

Members

All functions exposed in the stream module.

Functions

Function	Description
createArrayStream(Object) [page 601]	Creates a node.js readable stream by using a result set. The stream returns data as JavaScript arrays.

Function	Description
createLobStream(Object, Integer, Object) [page 602]	Creates a node.js readable stream by using a result set. The stream fetches data as LOB columns.
createObjectStream(resultset) [page 604]	Creates a node.js readable stream using a result set. The stream returns data as JSON objects.
createParameterLobStream(Statement, Integer, Object) Function [page 605]	Creates a readable stream by using an output LOB parameter.
createStatement(Connection, String[, Function]) Function [page 606]	Creates a statement object which executes a SQL statement with readable streams for input LOB parameters.

4.4.13.1 createArrayStream(ResultSet) Function

Creates a node.js Readable stream for a result set. The stream returns row data as arrays.

Syntax

```
createArrayStream(resultset)
```

Parameters

Type	Name	Description
ResultSet	`ResultSet object`	The result set returned by executing a query.

Returns

Returns a Readable stream object.

Remarks

The output stream returns row data formatted as arrays. Each row is an array consisting of all column values.

Example

Create a readable array stream and fetch data from it.

```
var hana = require('@sap/hana-client');
var conn = hana.createConnection();
conn.connect("serverNode=MyServer:30015;uid=system;pwd=manager");
var stmt = conn.prepare("SELECT * from M_SERVICES");
var rs = stmt.execQuery();
var hanaStream = require('@sap/hana-client/extension/Stream.js');
var stream = hanaStream.createArrayStream(rs);
var rows = [];
stream.on('readable', function (data) {
    var data;
    while (null !== (data = stream.read())) {
        rows.push(data);
    }
});
stream.on('end', function () {
    console.log(rows);
    rs.close();
    stmt.drop();
    conn.close();
});
```

4.4.13.2 createLobStream (ResultSet, Integer, Object) Function

Creates a node.js Readable stream for a result set. The stream returns row data as objects.

Syntax

```
createLobStream(resultset, columnIndex, options)
```

Parameters

Type	Name	Description
ResultSet	`resultset`	A result set.
Integer	`columnIndex`	The zero-based index of the LOB column.
Object	`options`	Properties that are specified as a JavaScript object. The only supported value is readSize. For example: `{readSize: 6000}`

Returns

Returns a Readable stream object.

Remarks

The output stream returns LOB column data as Buffer objects.

Example

The following example shows the total bytes fetched for the LOB.

> **Note**
>
> It would be much more efficient to get the LOB length with the `ResultSet.getValueLength` method.

```
'use strict';
var async = require('async');
var hana = require('@sap/hana-client');
var hanaStream = require('@sap/hana-client/extension/Stream');
var connOptions = {
    serverNode: 'myserver:30015',
    uid: 'system',
    pwd: 'manager'
};
var conn = hana.createConnection();
conn.connect(connOptions);
console.log('Create test table ......');
conn.exec('create table TEST_LOBS(ID integer primary key, DATA blob)');
console.log('Insert rows ......');
var stmt = conn.prepare('insert into TEST_LOBS values(?, ?)');
var buffer = null;
var ids = [1, 2, 3, 4, 5];
for (var i = 0; i < ids.length; i++) {
    buffer = new Buffer(new Array(1024 * 64 + ids[i]).join('ab'), "ascii");
    stmt.exec([ids[i], buffer]);
}
console.log('Fetching rows ......');
async.eachOfLimit(ids, ids.length, function (id, i, cb) {
    var stmt = conn.prepare('select ID, DATA from TEST_LOBS where ID = ?');
    var rs = stmt.execQuery([id]);
    rs.next(function (err, ret) {
        if (err) {
            console.log(err);
        } else {
            var id = rs.getValue(0);
            var lob = [];
            var totalBytes = 0;
            var lobStream = hanaStream.createLobStream(rs, 1,
               { readSize: 6000 });
            lobStream.on('data', function (buffer) {
                if (buffer) {
                    totalBytes += buffer.length;
                }
            });
```

```
            lobStream.on('end', function () {
                console.log(id + ', total bytes --- ' + totalBytes);
            });
            lobStream.on('error', function (error) {
                console.log(error);
            });
        }
    });
});
```

Related Information

ResultSet Class [page 567]

4.4.13.3 createObjectStream(ResultSet) Function

Creates a node.js Readable stream for a result set. The stream returns row data as objects.

Syntax

```
createObjectStream(resultset)
```

Parameters

Type	Name	Description
ResultSet	`resultset`	The ResultSet object returned by executing a query.

Returns

Returns a Readable stream object.

Remarks

The output stream returns row data formatted as objects. Each row object consists of a set of column name and value pairs for the current row. If multiple columns have the same column name, the property name for the second and subsequent columns with the same column name are "`<column-name>:<n>`" where `<n>` is a number starting from 1.

Example

Create a readable object stream and fetch data from it.

```
var hana = require('@sap/hana-client');
var conn = hana.createConnection();
conn.connect("serverNode=MyServer:30015;uid=system;pwd=manager");
var stmt = conn.prepare("SELECT * from M_SERVICES");
var rs = stmt.execQuery();
var hanaStream = require('@sap/hana-client/extension/Stream.js');
var stream = hanaStream.createObjectStream(rs);
var rows = [];
stream.on('readable', function (data) {
    var data;
    while (null !== (data = stream.read())) {
        rows.push(data);
    }
});
stream.on('end', function () {
    console.log(rows);
    rs.close();
    stmt.drop();
    conn.close();
});
```

4.4.13.4 createParameterLobStream(Statement, Integer, Object) Function

Creates a node.js Readable stream for a procedure call output LOB parameter.

Syntax

```
createParameterLobStream(statement, paramIndex, options)
```

Parameters

Type	Name	Description
Statement	`statement`	Specifies a statement.
Integer	`paramIndex`	Specifies a zero-based parameter index.
Object	`options`	Specifies properties as a JavaScript object. The only supported value is readSize. For example: `{readSize: 6000}`

Returns

Returns a Readable stream object.

Remarks

The output stream returns output LOB parameter data as Buffer objects. This is similar to `createLobStream`, except for a procedure call output LOB parameter instead of a result set LOB column.

4.4.13.5 createStatement(Connection, String[, Function]) Function

Creates a statement object which executes a SQL statement with readable streams for input LOB parameters.

Syntax

```
createStatement(connection, sql[, callback])
```

Parameters

Type	Name	Description
Connection	`connection`	Specifies the connection object.
String	`sql`	Specifies the SQL statement.
Function	`callback`	Specifies the optional callback function. The callback function is of the following form: `function(err, stmt){}`

Remarks

The created object has two methods:

Method	Description
`exec: function (params, callback)`	Executes the SQL statement. `params` specifies an optional array of binding parameters. You can add Readable streams to `params`. The callback parameter has the following form: `function(err, ret) {}`, where `ret` is the same return value as `Statement.exec[ute]`.
`drop: function (callback)`	Drops the statement.

Example

The following example creates a statement object and calls the returned statement object exec method with two Readable stream parameters.:

```
var hana = require('@sap/hana-client');
var hanaStream = require('@sap/hana-client/extension/Stream.js');
var Readable = require('stream').Readable;
var stream1 = new Readable();
stream1.push('blob1234');
stream1.push('blob5678');
stream1.push(null);
var stream2 = new Readable();
stream2.push('clob1234');
stream2.push('clob5678');
stream2.push(null);
var conn = hana.createConnection();
conn.connect("serverNode=myServer:30015;uid=system;pwd=manager");
conn.exec('CREATE TABLE HANA_NODE_TEST(ID INT, C_BLOB BLOB, C_CLOB CLOB)');
conn.setAutoCommit(false);
var sql = 'INSERT INTO HANA_NODE_TEST(ID, C_BLOB, C_CLOB) VALUES(?, ?, ?)';
hanaStream.createStatement(conn, sql, function (err, stmt) {
    if (err) {
        console.log(err);
    } else {
        stmt.exec([1, stream1, stream2], function (err, result) {
            if (err) {
                console.log(err);
            } else {
                console.log(result);
            }
            stmt.drop(function (err) {
                if (err) {
                    console.log(err);
                }
                conn.commit();
                conn.close();
            });
```

```
        });
    }
});
```

4.4.14 Promise Module

A Node.js module that exposes Promise-based versions of all asynchronous driver methods.

Syntax

```
var promiseModule = require('@sap/hana-client/extension/Promise.js');
```

Members

All functions exposed in the Promise module.

Functions

All functions in the tables below return a standard Node.js Promise object. All return values are handled using Promise resolution.

Connection Functions

Function	Description
abort(ConnObject)	Aborts the running database request that is being executed on ConnObject.
clearPool(ConnObject)	Empties the connection pool.
commit(ConnObject)	Performs a commit on ConnObject.
connect(ConnObject, {String | Object})	Creates a new connection to the database by using a connection string or a hash of connection parameters that is passed in as a parameter.
connect(ConnObject, Number)	Connects to the database by using an existing connection.
{disconnect | close | end}(ConnObject)	Closes the current connection.
exec[ute](ConnObject, String[,Array][,Object])	Executes the specified SQL statement.

Function	Description
prepare(ConnObject, String)	Prepares the specified SQL statement. The returned Promise resolves with a Statement object.
rollback(ConnObject)	Performs a rollback on the ConnObject.

For more information, see Connection Class [page 536].

Statement Functions

Function	Description
drop(StatementObject)	Drops the prepared SQL statement.
exec[ute](StatementObject[,Array][,Object])	Executes the prepared SQL statement.
exec[ute]Batch(StatementObject[,Array])	Executes the prepared SQL statement providing an array of parameters.
exec[ute]Query(StatementObject[,Array])	Executes the prepared SQL statement. The returned Promise resolves with a ResultSet object.
getData(StatementObject, Integer, Integer, Buffer, Integer, Integer)	Reads a stream of bytes from the specified output LOB parameter.
getParameterValue(StatementObject, Integer)	Retrieves output parameter values.
sendParameterData(StatementObject, paramIndex, buffer)	Sends LOB data to the server in chunks.

For more information, see Statement Class [page 583].

ResultSet Functions

Function	Description
close(ResultSetObject)	Closes the ResultSet object and frees up resources.
getData(ResultSetObject, Integer, Integer, Buffer, Integer, Integer)	Reads a stream of bytes from the specified LOB column.
getValues(ResultSetObject[,Options])	Gets an object with the column names and values of the current row.

Function	Description
next(ResultSetObject)	Gets the next row in the result set.
nextResult(ResultSetObject)	Moves to the next result set.

For more information, see ResultSet Class [page 567].

Example

Sample Code

```
var hana = require('@sap/hana-client');
var PromiseModule = require('@sap/hana-client/extension/Promise.js');
var conn = hana.createConnection();
var result;
PromiseModule.connect(conn,
"serverNode=myserver:30015;uid=system;pwd=manager")
  .then(() => {
       return PromiseModule.prepare(conn, "SELECT 'hello' AS RESULT FROM
DUMMY");
   })
   .then((stmt) => {
       return PromiseModule.executeQuery(stmt);
   })
   .then((rs) => {
       result = rs;
       return PromiseModule.next(result);
   })
   .then(() => {
       return PromiseModule.getValues(result);
   })
   .then((row) => {
       console.log(row);
       return PromiseModule.close(conn);
   })
   .catch(err => {
       console.log(err);
   });
```

4.5 ODBC Application Programming

Open Database Connectivity (ODBC) is a standard application programming interface (API) that provides a set of functions that enable applications to access a data model in a database. The SAP HANA client includes a dedicated ODBC interface.

4.5.1 Connect to SAP HANA via ODBC

SAP HANA provides a driver for connecting applications to the database with ODBC.

Context

Use the 32-bit ODBC driver for 32-bit applications, and the 64-bit driver for 64-bit applications.

Procedure

1. Install the ODBC driver. The driver is installed as part of the SAP HANA client installation.
2. Write code to create a connection to the database. You can use one of the following methods:

Connection String (SQLDriverConnect)

Use a connection string similar to the following (the command must be all on one line):

```
driver={<driver>};UID={<username>};PWD={<password>};
serverNode=<server>:<port>;databaseName=<dbname>
```

`<driver>` should be one of the following:

- HDBODBC (64-bit applications)
- HDBODBC32 (32-bit applications)

The following example shows a connection string that establishes a connection to an on-premise SAP HANA database server:

```
driver={HDBODBC};UID=myUser;PWD=myPassword;serverNode=myServer:30015
```

You can specify a specific database (for example, `TDB1`) using the `databaseName` parameter using the following on-premise example:

```
driver={HDBODBC};UID={myUser};PWD={myPassword};serverNode=myServer:30013;
databaseName=TDB1
```

> **Note**
>
> When using any of the characters: []{}(),;?*=!@ in a value, the value must be wrapped in curly braces, for example `{val;ue}`. If you are using the } character in a value (namely the username or password), then the application must format it correctly in the SQLDriverConnect string by specifying it twice, for example `{pass}}word}`.

`odbc.ini` file on Linux and UNIX (SQLDriverConnect or SQLConnect)

This file defines ODBC data sources on Linux and UNIX. User data sources are usually defined in `~/.odbc.ini` (where ~ is the user's home directory). However, the `~/.odbc.ini` location can be influenced by the ODBCINI environment variable. See Environment Variables [page 20] for more information. The `odbc.ini` file is used by the ODBC driver manager to find the ODBC driver and provide connection parameters. SAP HANA-specific connection parameter names are case sensitive. The following is an on-premise example data source in the `odbc.ini` file:

```
[HANADB1]
driver=/usr/sap/hdbclient/libodbcHDB.so
serverNode=myServer:30013
databaseName=DB1
DESCRIPTION=Sample HANA ODBC Data Source for DB1
```

You can use the following SQLDriverConnect connection string to connect using the above HANADB1 data source:

```
DSN=HANADB1;UID={myUser};PWD={myPassword}
```

> **Note**
>
> You cannot store UID or PWD values in the `odbc.ini` file.

ODBC Data Sources on Microsoft Windows (SQLDriverConnect or SQLConnect)

Create a data source by running the `odbcad32.exe` tool. You can run this tool on the command line or via the Control Panel by searching for "ODBC" and selecting "Set up ODBC data sources".

The Microsoft Windows ODBC Administrator (`odbcad32.exe`) tool is located in the `system32` directory. To create a data source for the 32-bit driver on a 64-bit Microsoft Windows machine, run the tool from the `SysWOW64` directory.

On the *DSN* tab, choose *Add*, select the HDBODBC or HDBODBC32 driver, and select *Finish*. In the resulting dialog, specify the name and details of the data source that you are adding:

Data Source Name

Specify a name for the ODBC data source.

Description

Provide a description for the ODBC data source.

Database Type

Choose one of the following database types:

- *SAP HANA Cloud*
  Host

  Specify the fully qualified domain name (FQDN) of the host.

(Optional) Provide the port number, `443`, at the end of the host name separated by a colon (:). If no port number is provided, the driver will append `:443` to the end of the host name. If any port number other than `443` is provided, the driver will replace it with `443`.

- *SAP HANA Database (Multitenant)*

  Host

Specify the fully qualified domain name (FQDN) of the host on which the system is installed.

Instance Number

Specify the instance number of the database to which you are connecting. The driver will convert the instance number to a port number when it is used to connect to the database.

Port Number

Specify the port number of the database to which you are connecting.

System Database

Choose *System Database* if the DSN is created to connect to SystemDB on the HANA server.

Tenant Database

Choose *Tenant Database* if the DSN is used to connect to the tenant database on the HANA server. Provide the tenant database name.

TLS/SSL

Select `Connect to the database using TLS/SSL` to encrypt communication between ODBC and the SAP HANA database by using the Transport Layer Security (TLS)/Secure Sockets Layer (SSL) protocol.

Optionally, specify the host name of the server in Override the hostname in the certificate. The value of this property must match the subject property of the certificate.

- *SAP HANA Database (Single-tenant)*

  Host

Specify the fully qualified domain name (FQDN) of the host on which the system is installed.

Instance Number

Specify the instance number of the database to which you are connecting. The driver will convert the instance number to a port number when it is used to connect to the database.

Port Number

Specify the port number of the database to which you are connecting.

Database

(Optional) Specify the database to which you are connecting.

TLS/SSL

Select `Connect to the database using TLS/SSL` to encrypt communication between ODBC and the SAP HANA database by using the Transport Layer Security (TLS)/Secure Sockets Layer (SSL) protocol.

Optionally, specify the host name of the server in Override the hostname in the certificate. The value of this property must match the subject property of the certificate.

If you used "HANADataSource" as the Data Source Name, then you can use the following SQLDriverConnection connection string to connect:

```
DSN=HANADataSource;UID={myUser};PWD={myPassword}
```

> **Note**
>
> You cannot store UID or PWD values in the Microsoft Windows DSN registry.

Related Information

Use the User Store (hdbuserstore) [page 856]
Connecting to SAP HANA Databases and Servers [page 8]

4.5.1.1 ODBC Connection Attributes

Use connection attributes to access client information with the ODBC application programming interface (API).

To set or get ODBC connection attributes that are specific to SAP HANA, first include `"<client install directory>/sdk/odbc/incl/sqlsdbodbc.h"`.

The following tables list the SAP HANA client-specific and standard ODBC connection attributes.

> **Note**
>
> The application receives the error message `Optional feature not implemented` if it tries to use any of the standard connection attributes not supported by the SAP HANA ODBC driver.

SAP HANA Client-Specific ODBC Connection Attributes

Connection Attribute	Type	Description
SQL_ATTR_AGGREGATE_TYPES	SQLULEN	Specifies aggregate types for expressions (AVG, SUM, and so on). 1 means like MS-SQL (for example, SUM(INTEGER) is of type INTEGER; otherwise, it would be FLOAT).
SQL_ATTR_BYTES_SENT	SQLBIGINT	Retrieves the total number of the network bytes sent to the server since the database connection was established. The total sent bytes are reset to zero by the driver when the connection is disconnected.
SQL_ATTR_BYTES_RECEIVED	SQLBIGINT	Retrieves the total number of the network bytes received from the server since the database connection was established. The total received bytes are reset to zero by the driver when the connection is disconnected
SQL_ATTR_CODEPAGE	SQLCHAR	Specifies the codepage used for UCS2 to ASCII conversion. The codepage is selected from the database. In Microsoft Windows, the default codepage is the active codepage of the client. In non-Microsoft Windows operating systems, the default codepage is WINDOWS-1252. A codepage is valid for all connections of the affected driver. If SQLSetConnectAttr is called a second time with a different codepage, then the driver can crash if a conversion is running simultaneously.
SQL_ATTR_DEFAULTMAPCHAR	SQLCHAR	Specifies the character representing non-convertible UCS2 characters in ASCII strings. This attribute only works if the SQL_ATTR_CODEPAGENAME attribute is set. If SQL_ATTR_CODEPAGENAME is not set, then a non-convertible UCS2 character causes a conversion error when converting to ASCII. Setting this attribute to '\0' disables this feature. The default setting is '?'.

Connection Attribute	Type	Description
SQL_ATTR_ENCRYPT	SQLULEN	Specifies whether encryption is used for the connection. Returns 0 if encryption is not used.
SQL_ATTR_TRACE_CALLBACK	ODBC_TraceCallback	Specifies a pointer to the callback method that receives connection trace messages.
SQL_ATTR_TRACE_CALLBACK_CONTEXT	SQLPOINTER	Specifies a context pointer passed directly to the callback.
SQL_ATTR_TRACE_CALLBACK_OPTIONS	SQLCHAR	Specifies a string describing the trace options to use for connection trace messages. See SQLDBC Trace Options [page 786]
SQL_ATTR_UNICODEDB	SQLULEN	Returns "SQL_TRUE" if the database supports Unicode, and "SQL_FALSE" otherwise (read-only).
SQL_SQLDBC_BINTEXT_IS_NCLOB	SQLULEN	Tells the ODBC driver to report BINTEXT fields as either NCLOB (default, TRUE) or LONG VARBINARY (FALSE).
SQL_SQLDBC_CHAR_AS_CESU8	SQLULEN	Specifies the encoding of SQL_C_CHAR data is in CESU8. This attribute cannot be set if SQL_SQLDBC_CHAR_AS_UTF8 is set.
SQL_SQLDBC_CHAR_AS_UTF8	SQLULEN	Specifies whether SQL_C_CHAR data is handled as UTF8. This attribute cannot be set if SQL_SQLDBC_CHAR_AS_CESU8 is set.
SQL_SQLDBC_GET_CLIENTINFO_VALUE	SQLCHAR	Retrieves the client information value for the key specified by the SQL_SQLDBC_SET_CLIENTINFO_KEY attribute.
SQL_SQLDBC_PROXY_HOST	SQLCHAR	Specifies the host name of the proxy server.
SQL_SQLDBC_PROXY_PORT	SQLULEN	Specifies the port of the proxy server. The port can be a value in the range of 1 and 65535. The default is 1080.

Connection Attribute	Type	Description
SQL_SQLDBC_PROXY_SCP_ACCOUNT	SQLCHAR	Provides SAP Cloud Connector routing information to the Cloud Connector SOCKS proxy. In an SAP BTP, Cloud Foundry Environment, provide an optional `<locationID>` in plaintext. Internally, the value is converted to base64 encoding. This value overrides the `proxyPassword` connection property. In an SAP BTP, Neo Environment, either `proxyScpAccount` or `proxyUsername` must be specified. `<subaccount>` is mandatory and `<locationID>` is optional. Both values are entered as plaintext strings and are internally converted to base64 encoding. This value overrides the `proxyUsername` connection property.
SQL_SQLDBC_PROXY_USERID	SQLCHAR	Specifies the proxy user name (for METHOD 02 authentication).
SQL_SQLDBC_PROXY_PASSWORD	SQLCHAR	Specifies the proxy password (for METHOD 02 authentication). When using the SAP HANA Cloud Connector, this field is used to provide the SAP HANA Cloud Connector location ID value.
SQL_SQLDBC_SET_CLIENTINFO_KEY	SQLCHAR	Sets the client information key that is used with the SQL_SQLDBC_GET_CLIENTINFO_VALUE or SQL_SQLDBC_SET_CLIENTINFO_VALUE attribute.
SQL_SQLDBC_SET_CLIENTINFO_VALUE	SQLCHAR	Sets the client information value for the key specified by the SQL_SQLDBC_SET_CLIENTINFO_KEY attribute.
SQL_SQLDBC_SHOW_CATALOGS	SQLULEN	Tells the ODBC driver whether SQL catalogs are visible. The default is FALSE.

Connection Attribute	Type	Description
SQL_SQLDBC_SPATIALTYPES	SQLULEN	Specifies the spatial type returned for ST_POINT and GT_GEOMETRY columns. This attribute can be specified as follows: • 2 = SQL_TYPE_DEST_GEOMETRY (ODBC default) • 1 = SQL_TYPE_ST_POINT(75) / SQL_TYPE_ST_GEOMETRY(74) • 0 = SQL_VARBINARY(-3)

ODBC Standard Connection Attributes Supported by the SAP HANA ODBC Driver

Connection Attribute	Type	Description
SQL_ATTR_CURRENT_CATALOG	SQLCHAR	Retrieves the database name of the current connection. This attribute cannot be changed.
SQL_ATTR_TXN_ISOLATION	SQLUINTEGER	Specifies the transaction isolation level for the connection. The transaction isolation level can be set as follows: • 0 = Read uncommitted • 1 = Read committed • 2 = Repeatable read • 3 = Serializable The default transaction isolation level is 1.

Connection Attribute	Type	Description
SQL_ATTR_AUTOCOMMIT	SQLUINTEGER	Specifies the auto commit mode. The auto commit mode can be set as follows: • 0 = SQL_AUTOCOMMIT_OFF • 1 = SQL_AUTOCOMMIT_ON With SQL_AUTOCOMMIT_OFF the driver uses manual commit mode. The application must explicitly commit or roll back transactions with SQLEndTran. With SQL_AUTOCOMMIT_ON the driver uses auto commit mode. Each statement is committed immediately after it is executed. This is the default. Any open transactions on the connection are committed when SQL_ATTR_AUTOCOMMIT is set to SQL_AUTOCOMMIT_ON.
SQL_ATTR_LOGIN_TIMEOUT	SQLUINTEGER	Specifies the login timeout. The value corresponds to the number of seconds to wait for a login request to complete before returning to the application. The driver returns an error back to the application if it cannot complete the login within a given time. The default setting is 0. That means the timeout is disabled and a connection attempt will wait indefinitely.
SQL_ATTR_TRACE	SQLUINTEGER	Specifies ODBC tracing. The ODBC tracing can be set as follows: • SQL_OPT_TRACE_OFF • SQL_OPT_TRACE_ON The default setting is SQL_OPT_TRACE_OFF. When SQL_ATTR_TRACE is set to SQL_OPT_TRACE_ON the Driver Manager writes each ODBC function call to the trace file.

Connection Attribute	Type	Description
SQL_ATTR_TRACEFILE	SQLCHAR	Specifies the ODBC tracing file. The ODBC driver manager writes ODBC function calls to the specified tracing file if SQL_ATTR_TRACE is set to SQL_OPT_TRACE_ON.
SQL_ATTR_PACKET_SIZE	SQLUINTEGER	Specifies the network packet size in bytes. The default packet size is 1 MB.

Remarks

ODBC applications can use the connection attributes, SQL_SQLDBC_SET_CLIENTINFO_KEY and SQL_SQLDBC_SET_CLIENTINFO_VALUE to set the client info key/value pairs to the server using the ODBC function, SQLSetConnectAttr. The driver manager caches the attributes if they are set before calling the SQLDriverConnect or SQLConnect and applies the attributes to the ODBC driver after it is loaded. If the key/value pairs are APPLICATION ID and APPLICATIONUSER, these key/value pairs must be set before calling the SQLDriverConnect or SQLConnect function. How to cache and apply the attributes depends on the individual ODBC driver manager. There are some limitations regarding to these connection attributes on Microsoft Windows with the Microsoft ODBC driver manager and on UNIX with the unixODBC driver manager.

Microsoft Windows:

The driver manager only caches one set of the key/value pairs and the last set is applied to the ODBC driver. Therefore, only one of the APPLICATION ID or APPLICATIONUSER can be set through SQLSetConnectAttr per connection. If the application requires both the APPLICATION ID and APPLICATIONUSER, they must be supplied through the connection string on Microsoft Windows.

unixODBC:

The unixODBC driver caches all the client info key/value pairs if they are set before calling SQLDriverConnect or SQLConnect, but the driver manager applies the attributes to the driver in reverse order. Therefore, the application must call SQLSetConnectAttr with the attribute, SQL_SQLDBC_SET_CLIENTINFO_VALUE first, then call the API for the SQL_SQLDBC_SET_CLIENTINFO_KEY for each pair of client info key/values.

Related Information

Setting Session-Specific Client Information [page 16]
SQLDBC Trace Options [page 786]

4.5.1.2 ODBC Connection Properties

When connecting to an SAP HANA database using ODBC, you can use ODBC connection properties and SQLDBC connection properties.

> **Note**
>
> - Do not set connection properties to an empty string. The client will substitute any empty string values with the property's default setting.
> - The connection properties can be specified on a DSN and a connection string at the same time. If the same connection property is specified on both, the ODBC driver will take the property setting from the connection string and ignore the one from the DSN.
> - When specifying connection property values in a connection string to `SQLDriverConnect`, specify values containing the following characters between curly braces: `{ <value> }`
>   - `[ ] { } ( ) , ; ? * = ! @`
>   - Note that any `}` characters that appear in `<value>` must be specified twice: `{ }} }`

The following table lists the SAP HANA client-specific ODBC connection properties (property names are case insensitive).

Property	Value	Default	Description
`allowFetchWarnings`	Boolean	TRUE	Controls whether server warnings from retrieving the result set of a query is returned to an ODBC application when it calls ODBC APIs SQLFetch or SQLExtendedFetch. When this property is set to be TRUE (default), these two APIs can return SQL_SUCCESS_WITH_INFO, if a server warning occurs during result set retrievals.
`bintext_is_nclob`		TRUE	Determines which SQL type is used to describe BINTEXT columns. TRUE if the type is returned as SQL_WLONGVARCHAR(-10), and FALSE if the type is returned as SQL_LONGVARBINARY(-4). Use FALSE to maintain compatibility with SAP HANA versions earlier than 2.0.
`char_as_utf8`	Boolean	FALSE	Returns CHAR data in UTF-8 format.

Property	Value	Default	Description
`driver`			Specifies the driver information to use for the ODBC connection. On Microsoft Windows, this is a name that refers to the driver name that corresponds to the `ODBCINST.INI` registry key. On all other platforms, this is a path to the shared object for the SAP HANA ODBC driver. For example, `driver=/usr/sap/hdbclient/libodbcHDB.so`.
`DSN`			Specifies the ODBC driver entry to use for the driver information and other connection properties that are managed by the ODBC driver manager.
`enhancedWarnings`			Offers additional warning information when explicitly enabled after an ODBC operation. Successful operations return SQL_SUCCESS_WITH_INFO instead of SQL_SUCCESS when there are warnings available from the SQLGetDiagRec function. Failed operations still receive errors at the first record from SQLGetDiagRec and there may be additional warning records made available. In the case of a failed operation, the return code of the operation remains SQL_ERROR, regardless of additional warnings. The list of warnings are mostly in chronological order.

Property	Value	Default	Description
PWD	<password>		Specifies the user password. Optional, depending on the used authentication method.
serverNode	<host>:<port>[, <host>:<port>]		Specifies the SAP HANA server address or addresses. If your system contains a failover standby node, you can specify it as an additional optional address. Enclose the values between curly braces when specifying multiple host addresses.
show_catalogs	Boolean		Enables Microsoft-specific application support. This is needed to connect to SAP HANA from Microsoft Excel and Microsoft Access. The default setting for this property is TRUE on Microsoft Windows, if the application name is Microsoft Access or Excel. Otherwise, the default setting is FALSE.
spatialTypes	<number> or Boolean	2	Determines the ODBC SQL type returned for the ST_POINT and ST_GEOMETRY columns. Possible values are: • 0 or FALSE - returns SQL_VARBINARY • 1 or TRUE - returns SQL_TYPE_ST_POINT and SQL_TYPE_ST_GEOMETRY • 2 - returns SQL_TYPE_DST_GEOMETRY

Property	Value	Default	Description
`UID`	`<user>`		Specifies the user name. This property is optional depending on the used authentication method. The alternate forms are `Username` and `UserID`.

SQLDBC Connection Properties

The following table lists SQLDBC connection properties (property names are case insensitive) that can also be used with ODBC:

> **Note**
>
> A valid setting for a Boolean property can be: “1”, “TRUE”, “YES”, or “ON” for a TRUE setting or “0”, “FALSE”, “NO”, or “OFF” for a FALSE setting. The setting of a Boolean property is case-insensitive.

Property	Value	Default	Description
`abapVarCharMode`	Boolean	FALSE	Specifies that in ABAP VARCHAR mode, a single space and an empty string are considered the same.

Property	Value	Default	Description
`allowLocalCompress`	Boolean	TRUE	Determines whether or not connections from the same machine as the SAP HANA server are compressed. The `compress` parameter and/or the `indexserver.ini` > *session* > *compression* server configuration control whether or not network compression is enabled. **Note** If the connection uses a proxy, do not use `allowLocalCompress`=FALSE as the connection may not be compressed when required.
`allowReconnectOnSelect`	Boolean	OFF	Allows reconnects to happen for SELECT queries which fail during a receive. By default, this property is set to OFF, which allows for backward compatibility.

Property	Value	Default	Description
`bindAddress`	`<ip-address>`	NULL	Specifies that the provided address is used as the source address for the socket connection. By default, the address preferred by the OS is used. If setting `bindAddress` to a hostname, that hostname should only resolve to a single IP address. In the majority of network configurations, specifying `bindAddress` is not required. **Note** Only specify `bindAddress` if it is required to connect in your network configuration.
`chopBlanks`	Boolean	FALSE	Removes trailing blanks from character output.
`chopBlanksInput`	Boolean	FALSE	Removes trailing blanks from character input.
`communicationTimeout`	Timeout in milliseconds	0 (disabled)	Aborts communication attempts to the server after the specified timeout. If set to a non-zero value, the driver will wait for a maximum of the specified number of milliseconds for any request to the server to complete. If the time has expired and a response has not been received, the driver returns an error.

Property	Value	Default	Description
`compress`	Boolean	FALSE	Enables or disables network compression for the connection. The `indexserver.ini` > *session* > *compression* server configuration defaults to FALSE. Setting this parameter overrides the setting of the `indexserver.ini` > *session* > *compression* parameter. The `allowLocalCompress` property controls if clients running on the same machine as the server use network compression when compression is requested.
`connDownRollbackError`	Boolean	FALSE	Setting this property to TRUE specifies that if the database connection is down due to network or SAP HANA server problems, the client rolls back the last transaction and sends and error to the application. The application can resend the last transaction to the server after properly handling the error. The following conditions must be true: • The session is recoverable in the server • There is a written transaction (uncommitted operation) in the session • The client failed to send the last SQL statement, or the last statement was sent to the server successfully, but both auto-commit and DDL auto-commit in the client are off. By default, this property is set to FALSE.

Property	Value	Default	Description
`connectTimeout`	Timeout in milliseconds	0 (limited by the `nodeConnectTimeout` connection property)	Aborts connection attempts after the specified timeout. `connectTimeout` limits the overall time for the entire connection to try all server addresses specified. See `nodeConnectTimeout` for per-node timeout.
`currentSchema`	`<schema>`	CURRENTUSER	Sets the current schema, which is used for identifiers without a schema.
`cursorHoldabilityType`	NONE, COMMIT, ROLLBACK, COMMIT_AND_ROLLBACK	COMMIT	Specifies whether the cursor for a result set remains open after the transaction is committed or rolled back. If a cursor remains open, then the application is able to fetch more rows from it. NONE means the cursor is always closed after a commit or rollback, while COMMIT means the cursor remains valid after a commit, but not after rollback. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources. Support for holding a cursor over ROLLBACK or COMMIT_AND_ROLLBACK is available in SAP HANA servers beginning with SAP HANA 2.4.

Property	Value	Default	Description
`emptyTimestampIsNull`	Boolean	TRUE	When enabled, DATE, TIME, SECONDDATE, and TIMESTAMP values inserted as empty strings are returned as NULLs. When disabled, these values are returned as the following out-of-band values: • `DATE` - one day before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 00:00.000000000 UTC BC • `TIME` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `SECONDDATE` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `TIMESTAMP` - one hundred nanoseconds before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.999999900 UTC BC

Property	Value	Default	Description
`ignoreTopology`	Boolean	FALSE	Controls how the client uses the topology returned from the server. TRUE indicates to use the topology unless port-forwarding is detected and FALSE indicates to always ignore the topology. It is an error to specify both ignoreTopology=TRUE and siteType=PRIMARY or SECONDARY.
`isolationLevel`	TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE	TRANSACTION_READ_COMMITTED	Sets the isolation level for the connection.
`isoTimestampOutput`	`<string>`	FALSE	Displays whether or not 'T' is used as the separator between the Date and Time parts of a timestamp retrieved as a string: TRUE/FALSE. If FALSE, then a space character is used as the separator.
`key`	`<username>`	Empty	Specifies the key for the hdbuserstore.
`maxLazyDroppedStatements`	`<number>`	50	Controls the maximum number of queued statements to be dropped. Setting this value to 0 disables the lazy drop feature and causes a request to be sent to the server each time a statement is dropped by the application.
`maxVersion`	`<string>`		Specifies the maximum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.

Property	Value	Default	Description
`minVersion`	`<string>`		Specifies the minimum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.
`nodeConnectTimeout`	Timeout in milliseconds	60000	Aborts the connection attempt after the specified maximum timeout. The connection attempt typically fails immediately if the server cannot be found. If connection attempts to a particular server address are slow to fail, then using this connection property reduces the time before a successful connection by minimizing the time waiting to connect to individual addresses. On a host with multiple resolved IP addresses, the network timeout for each attempt to connect to an IP address will be equal to the `nodeConnectTimeout` divided by the number of resolved IP addresses. If this option is set to zero, the `connectTimeout` setting will be used. If both of the `nodeConnectTimeout` and `connectTimeout` are set to zero, then SQLDBC will use the system TCP/IP socket connect timeout to connect to the IP address.

Property	Value	Default	Description
`packetCaching`	Boolean	TRUE	Caching packets reduces the need to continually allocate and deallocate memory for packets. However, this increases the peak memory usage of the application, and for how long it stays at or near that peak.
`packetSize`	`<number>` in bytes	1048576 (1 MB)	Sets the maximum size of a request packet sent from the client to the server, in bytes. In some cases, request packets can be larger (for example, if a SQL statement or parameter value is larger than the packetSize). The minimum is 1 MB.
`packetSizeLimit`	`<number>` in bytes	2147483647 (2 GB)	Sets the maximum size for a request or reply packet, in bytes. If SQLDBC needs to allocate a packet larger than the limit, then it returns an error. If a reply packet is too large, then the connection is dropped. The minimum value is the packetSize value; the maximum value is 2 GB.

Property	Value	Default	Description
`prefetch`	Boolean	TRUE	Controls result set prefetching. When enabled, result set prefetch requests the next block of rows of a result set while the application is still processing the current block of rows. When fetching large result sets, result set prefetch can improve performance; however, it can also result in the client using more memory. In some cases, the server JobExecutor thread could be blocked from sending the reply for an indefinite period of time until the client receives the prefetch reply. If the result set contains LOB columns, then result set prefetch is always disabled.
`{ proxyHostname | proxy_host }`	`<host-name>`		Specifies the host name of the proxy server. The default proxy server is SOCKS5. If `proxyHttp` is set to true, the proxy server is assumed to be an HTTP proxy.
`proxyHttp`	Boolean	FALSE	When set to TRUE, enables HTTP proxy authentication and implies that ENCRYPT=TRUE. This property is only applicable if `proxyHostname` is specified and `webSocketURL` is not specified.

Property	Value	Default	Description
`{ proxyPassword | proxy_password }`	`<password>`		Specifies the password for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the Cloud Connector location ID value. The value must be entered as a base64 encoded value with a "1." prepended to the string.
`{ proxyPort | proxy_port }`	`<port-number>`		Specifies the port of the proxy server. The default is 1080.
`{ proxyScpAccount | proxy_scp_account }`	SAP BTP, Cloud Foundry Environment: `<location-ID>` SAP BTP, Neo Environment: `<subaccount>.<location-ID>`		Provides SAP Cloud Connector routing information to the Cloud Connector SOCKS proxy. In an SAP BTP, Cloud Foundry Environment, provide an optional `<locationID>` in plaintext. Internally, the value is converted to base64 encoding. This value overrides the `proxyPassword` connection property. In an SAP BTP, Neo Environment, either `proxyScpAccount` or `proxyUsername` must be specified. `<subaccount>` is mandatory and `<locationID>` is optional. Both values are entered as plaintext strings and are internally converted to base64 encoding. This value overrides the `proxyUsername` connection property.

Property	Value	Default	Description
`{ proxyUserName | proxy_userid }`	`<user-name>`		Specifies the user name for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the JSON Web Token (JWT) value. When using an SAP HANA Cloud Connector in the SAP BTP, Neo Environment, this field is used to provide the subaccount and location ID details. The values must be base64 encoded manually using the format `1.base64(subaccount).base64(locationID)`.
`reconnect`	Boolean	TRUE	Controls whether the system automatically reconnects to the database instance after a command timeout or when the connection is broken. Reconnecting restores the old state (for example, if no transaction was open).

Property	Value	Default	Description
`resolveHostName`	IPv4First IPv6First	IPv4First	Controls which addresses to use in a server address, especially if an address or hostname resolves to multiple IP addresses. In all cases, stop attempting further resolved addresses once a TCP/IP connection is successfully established. The following values can be used with `resolveHostName`: • IPv4First (default) - try all resolved addresses starting with IPv4 addresses. • IPv6First - try all resolved addresses starting with IPv6 addresses.
`sessionVariable:<key-name>`	`<session variable value>`	None except for APPLICATION and APPLICATIONUSER keys	Sets the session variable `<key-name>` to `<session variable value>`. For example, to set the session variable with key `APPLICATION`, use `sessionVariable:APPLICATION=MyApplication`.

Property	Value	Default	Description
`statementCacheSize`	`<number>`	0 (no statements cached)	Controls the maximum number of cached prepared statements. Caching SQL statements that are frequently prepared and dropped can improve performance. DDL can cause behavior changes when statement caching is enabled. **Note** If it is possible that the definition of objects referred to by the prepared statements may change because DDL is executed while you are connected, then it is recommended that you do not enable prepared statement caching.
`statementRoutingFailureBackoff`	Boolean	TRUE	Enables or disables the re-establishment of an internal connection to a specific host for a time period, after the internal connection fails during the preparation or execution of a statement routed to that host. The minimum time period is 1 minute. The time period is doubled after each subsequent internal connection establishment or usage failure for the same node, up to a maximum of 1 hour. Every successful internal connection re-establishment and usage resets the time period to the minimum value.

Property	Value	Default	Description
`statementRoutingWarnings`	Boolean	FALSE	Returns a SQL warning to the application with SQLCODE -11001 or -11003 that gives information about the routing failure of a prepared statement.
`tcpKeepAliveCount`	`<number-of-retries>` This value is ignored if it is 0.	5	Controls the number of KeepAlive packets sent to the server before the driver assumes that the connection is closed. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.
`tcpKeepAliveIdle`	`<seconds>` This value is ignored if it is 0.	200	Controls the number of seconds that the connection is idle before the driver sends a KeepAlive packet. If 0 is specified, this value is determined by the operating system.
`tcpKeepAliveInterval`	`<seconds>` This value is ignored if it is 0.	4	Controls the number of specified seconds before the driver sends an additional KeepAlive packet if there was no response to the previous KeepAlive packet. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.

Property	Value	Default	Description
`timestampPadding`	Boolean	TRUE	In a timestamp where the format contains a fraction, this property determines whether the fractional portion of the timestamp will be 7 or 9 digits. When disabled, the fractional portion is 7 digits to match how the server displays timestamps. Otherwise, it is 9 digits. **Note** ABAP timestamps are not affected regardless of the value of `timestampPadding`.
`traceFile`			Specifies the trace file name. Only the stdout and stderr values are supported.
`traceOptions`			Implicitly enables tracing for all SQLDBC-based client connections. See the HDB_SQLDBC_TRACEOPTS description in the **Environment Variables** topic for more information.
`webSocketURL` (SAP HANA Service only)	`<webSocketUrl>`		Specifies the WebSocket URL used to connect to a SAP HANA server. The connection property value cannot be empty. WebSockets connections have the following default connection property settings: • encrypt=TRUE • compress=TRUE • reconnect=FALSE

Security Connection Properties

Property	Value	Default	Description
`{ cseKeyStorePassword | clientside_encryption_keystore_password }`	`<string>`		Provides the password for the local key store. This option is required when using client-side encryption. Passwords must be at least eight characters and have at least one upper case character, one lower case character, and one number.
`authenticationMethods`	`<comma-separated-string>`		Provides a comma-separated list of authentication methods to be used by the client. Possible values are: password, pbkdf2, ldap, saml, jwt, saplogon, kerberos, sessioncookie, and X509.
`authenticationX509`	`<string>`	`<value-of-sslKeyStore>`	Provides a string containing the X509 certificate or the name of a file containing the X509 certificate. The certificate must be PEM-encoded in PKCS8 format. Defaults to the existing value of `sslKeyStore`, if specified.
`authenticationX509Password`	`<password>`		With CommonCryptoLib, specifies the PIN for the PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`encrypt`	Boolean	FALSE	Enables or disables TLS encryption. For SAP HANA versions 2.5 and lower, set encrypt=TRUE when connecting to port 443 (SAP HANA Cloud). For SAP HANA versions 2.6 and higher, encrypt=TRUE is not required when connecting to port 443.

Property	Value	Default	Description
`sslCryptoProvider`	{ commoncrypto | openssl | mscrypto }	1. commoncrypto (if installed) 2. openssl/mscrypto	Specifies the cryptographic library provider used for TLS communication. If you specify a value for this property, then you must also explicitly specify paths in both the `sslKeyStore` and `sslTrustStore` properties to avoid configuration issues. If CommonCryptoLib is not available, OpenSSL is used by default in Linux environments, and msCrypto in Microsoft Windows environments. OpenSSL is not available on Microsoft Windows. **Note** Check the `client` folder of the installation package to see if `COMMONCRYPTOLIB.TGZ` is present. If not, you can download CommonCryptoLib separately. For instructions on how to download CommonCryptoLib, see "Download and Install SAP Common Crypto Library" in the *SAP HANA Client Installation and Update Guide*.
`sslHostNameInCertificate`	`<string>`	Empty	Specifies the host name used to verify server's identity. The host name specified here verifies the identity of the server instead of the host name with which the connection was established. For example, in a single-host system, if a connection is established from a client on the same host as the server, then a mismatch would arise between the host named in the certificate (actual host name) and the host used to establish the connection (localhost). If you specify `*` as the host name, then the server's host name is not validated. Other wildcards are not permitted.

Property	Value	Default	Description
`sslKeyStore`	`<file>` | `<PEM-encoded-identity>` msCrypto: MY | Root | Trust | CA	`$SECUDIR/sapcli.pse` (CommonCryptoLib) `$HOME/.ssl/key.pem` (OpenSSL) MY (msCrypto)	Specifies the path to the keystore file that contains the client's identity. An identity is used for mutual TLS authentication and consists of the client's private key, the client's certificate, and, optionally, the certificate of the signing authority that signed the client's certificate. If you are using CommonCryptoLib, then the PSE file must also contain the trust store (for example, the server's public certificates) and the sslTrustStore property should be empty. If you are using CommonCryptoLib, but not doing mutual TLS authentication, then the PSE file contains only the trust store. If you are using CommonCryptoLib, then use the SAPGENPSE tool (installed with CommonCryptoLib) to create the client PSE (`sapcli.pse`) and generate the client's keys. You must also import the server's public certificates into `sapcli.pse`. Typically, this is the root certificate or the certificate of the certification authority that signed the server's public certificates. See *SAP Note 1718944 (SAP HANA DB: Securing External SQL Communication (CommonCryptoLib))*. If you are using OpenSSL, use OpenSSL tools to create the required keystore file. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.

Property	Value	Default	Description
`sslMinProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	TLS12	Controls the minimum TLS version. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The lowest protocol version is accepted in TLS handshakes.
`sslMaxProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	MAX	Controls the maximum TLS version. MAX allows this parameter to be future-proof in the event that new TLS versions are supported. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The highest negotiated protocol version is used in TLS handshakes.
`sslKeyStorePassword`	`<password>`		With CommonCryptoLib, specifies the PIN for the keystore PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`sslSNIHostname`	`<string>`	Empty	Specifies the name of the host that is attempting to connect at the start of the TLS handshaking process.
`sslSNIRequest`	Boolean	TRUE	Specifies whether SNI requests are enabled for TLS connections: TRUE/FALSE.

Property	Value	Default	Description
`sslTrustStore`	`<file>` | `<PEM-encoded-trust-store>` msCrypto: MY | Root | Trust | CA	`$HOME/.ssl/trust.pem` (OpenSSL) sapcli.pse (CommonCryptoLib) MY (msCrypto)	Specifies the path to a trust store file that contains the server's public certificates if using OpenSSL. Typically, the trust store contains the root certificate or the certificate of the certification authority that signed the server's public certificates. If you are using the cryptographic library CommonCryptoLib or msCrypto, leave this property empty. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.
`sslUseDefaultTrustStore`	Boolean	TRUE	(Only applies to the OpenSSL Crypto Provider) Set to FALSE in order to disable the use of the default Certificate Authorities provided with OpenSSL or the Operating System.
`sslValidateCertificate`	Boolean	TRUE	Specifies whether to validate the server's certificate.

Related Information

Download and Install SAP Common Crypto Library
Setting Session-Specific Client Information [page 16]

4.5.1.2.1 How To Set ODBC Connection Properties

In this example, the packet size is changed to 2MB by passing "packetSize=2097152" as part of the connection string.

The following example shows how to set ODBC connection properties:

```
SQLHENV henv;
```

```
SQLHDBC hdbc;
SQLRETURN rc;
SQLCHAR outString[1024];
SQLSMALLINT outLength;
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
rc = SQLSetEnvAttr( henv, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
rc = SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc );
rc = SQLDriverConnect( hdbc, henv,
   (SQLCHAR
*)"driver={HDBODBC};UID={myUser};PWD={myPassword};serverNode=myServer:30015;packe
tSize=2097152",
   SQL_NTS, outString, sizeof(outString), &outLength,
SQL_DRIVER_NOPROMPT);
```

4.5.2 ODBC Statement Attributes

The ODBC driver supports statement attributes that define characteristics of a statement.

The following tables list the SAP HANA client-specific and standard ODBC statement attributes.

> **Note**
>
> The application receives the error message `Option type out of range` if it tries to use any of the standard statement attributes not supported by the HANA ODBC driver.

SAP HANA Client-Specific ODBC Statement Attributes

Statement Attribute	Type	Description	Example

SQL_ATTR_PRINTLINE	SQLCHAR	Retrieves the next print line from the SQLSCRIPT_PRINT library.
		When no lines remain, SQLGetStmtAttr returns SQL_NO_DATA. If the provided buffer is not large enough to store the line, then it is truncated. You cannot repeat the call to retrieve the untruncated line. To avoid truncation, provide a buffer with sufficient space to store 32767 characters.
			`SQLExecDirect(hstmt, (SQLCHAR*)`
			`"CREATE PROCEDURE PrintAPITest AS "`
			`"BEGIN "`
			`"USING SQLSCRIPT_PRINT AS PRINT; "`
			`"PRINT:PRINT_LINE('Hello, world'); "`
			`"END ", SQL_NTS);`
			`SQLExecDirect(hstmt, (SQLCHAR*) "CALL PrintAPITest", SQL_NTS);`
			`SQLCHAR buf[64];`
			`SQLINTEGER ind;`
			`SQLGetStmtAttr(hstmt, SQL_ATTR_PRINTLINE, &buf, sizeof(buf), &ind);`
			`puts(buf);`
			`SQLExecDirect(hstmt, (SQLCHAR*) "DROP PROCEDURE PrintAPITest", SQL_NTS);`

SQL_ATTR_QUERY_TIMEOUT	SQLINTEGER	Sets the number of seconds to wait for an SQL statement to execute.	The following example sets 0 as the number of seconds to wait before the timeout occurs: `SQLSetStmtAttr( stmt, SQL_ATTR_QUERY_TIMEOUT, 0, SQL_IS_UINTEGER );`

Statement Attribute	Type	Description	Example
SQL_ATTR_SERVER_CPU_TIME	SQLBIGINT	Retrieves the total CPU processing time, in microseconds, on the server side, for the statement prepare, execute, or fetch request. This value does not include network communication.	`SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_CPU_TIME, &cpuTime, SQL_IS_INTEGER, NULL);`
SQL_ATTR_SERVER_MEMORY_USAGE	SQLBIGINT	Retrieves the memory usage on the server side, for the statement prepare, execute, or fetch request, in bytes.	`SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_MEMORY_USAGE, &memoryUsage, SQL_IS_INTEGER, NULL);`
SQL_ATTR_SERVER_PROCESSING_TIME	SQLBIGINT	Retrieves the elapsed processing time, in microseconds, on the server side, for the statement prepare, execute, or fetch request. This value does not include network communication.	`SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_PROCESSING_TIME, &processingTime, SQL_IS_INTEGER, NULL);`

SQL_ATTR_STMT_PACKET_SIZE	SQLINTEGER	Sets the network packet size for the statement, in bytes.
		The value should be greater than 1 MB and less than 2 GB, but a value less than 100 MB is recommended.
			`SQLHSTMT hstmt;`
			`SQLINTEGER sPacketSize = 2 * 1024 * 1024;`
			`SQLINTEGER gPacketSize;`
			`// allocate a statement handle`
			`SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);`
			`// set the statement packet size`
			`SQLSetStmtAttr(hstmt, SQL_ATTR_STMT_PACKET_SIZE, (SQLPOINTER)sPacketSize, SQL_IS_INTEGER);`
			`// retrieve the statement packet size`
			`SQLGetStmtAttr(hstmt, SQL_ATTR_STMT_PACKET_SIZE, &tPacketSize, SQL_IS_INTEGER, NULL);`

ODBC Standard Statement Attributes Supported by the HANA ODBC Driver

Statement Attribute	Type	Description	Example
SQL_ATTR_CONCURRENCY	SQLULEN	Specifies the cursor concurrency. This attribute can be specified as follows: • SQL_CONCUR_READ_ONLY: Cursor is read-only, and no updates are allowed. This is the default. • SQL_CONCUR_LOCK: Cursor uses the lowest level of locking sufficient to ensure that the row can be updated.	`SQLSetStmtAttr(hstmt, SQL_ATTR_CONCURRENCY, (SQLPOINTER)SQL_CONCUR_READ_ONLY SQL_IS_UINTEGER);`
SQL_ATTR_CURSOR_SCROLLABLE	SQLULEN	Specifies the characteristics of a cursor. This attribute can be specified as follows: • SQL_NONSCROLLABLE: Scrollable cursors are not required. If the application calls SQLFetchScroll on this handle, the only valid value of FetchOrientation is SQL_FETCH_NEXT. This is the default. • SQL_SCROLLABLE: Scrollable cursors are required. When calling SQLFetchScroll, the application may specify any valid value of FetchOrientation, achieving cursor positioning in modes other than the sequential mode.	`SQLSetStmtAttr(hstmt, SQL_ATTR_CURSOR_SCROLLABLE, (SQLPOINTER)SQL_SCROLLABLE, SQL_IS_UINTEGER);`

Statement Attribute	Type	Description	Example
SQL_ATTR_CURSOR_SENSITIVITY	SQLULEN	Specifies whether cursors for the specified statement make visible the changes made to a result set by another cursor. This attribute can be specified as follows: • SQL_UNSPECIFIED: It is unspecified what the cursor type is and whether cursors on the statement handle make visible the changes made to a result set by another cursor. This is the default. • SQL_INSENSITIVE: All cursors on the statement handle show the result set without reflecting any changes made to it by any other cursor. • SQL_SENSITIVE: All cursors on the statement handle make visible all changes made to a result set by another cursor.	`SQLSetStmtAttr(hstmt, SQL_ATTR_CURSOR_SENSITIVITY, (SQLPOINTER) SQL_INSENSITIVE, SQL_IS_UINTEGER);`
SQL_ATTR_CURSOR_TYPE	SQLULEN	Specifies the cursor type. This attribute can be specified as follows: • SQL_CURSOR_FORWARD_ONLY: The cursor only scrolls forward. This is the default. • SQL_CURSOR_STATIC: The data in the result set is static.	`SQLSetStmtAttr(hstmt, SQL_ATTR_CURSOR_TYPE, (SQLPOINTER) SQL_CURSOR_FORWARD_ONLY, SQL_IS_UINTEGER);`

Statement Attribute	Type	Description	Example
SQL_ATTR_MAX_ROWS	SQLULEN	Specifies the maximum number of rows to return to the application for a query. If the maximum number of rows is set to 0, the driver returns all rows. The default maximum number of rows is 0.	`SQLSetStmtAttr(hstmt, SQL_ATTR_MAX_ROWS, (SQLPOINTER)maxRows, SQL_IS_UINTEGER);`
SQL_ATTR_PARAM_BIND_OFFSET_PTR	SQLULEN	Specifies an offset added to pointers to change binding of dynamic parameters. If this field is set to a non-null integer array, the driver dereferences the pointer, adds the dereferenced value to each of the deferred fields in the descriptor record, and uses the new pointer values when binding. This attribute is set to null by default.	`SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_BIND_OFFSET_PTR, offsetValue, SQL_IS_POINTER);`
SQL_ATTR_PARAM_BIND_TYPE	SQLULEN	Specifies the binding orientation used for dynamic parameters. The default setting for this field is SQL_PARAM_BIND_BY_COLUMN. The driver chooses column-wise binding when the default is set.	`SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_BIND_TYPE, SQL_PARAM_BIND_BY_COLUMN, SQL_IS_UINTEGER);`
SQL_ATTR_PARAM_OPERATION_PTR	SQLUSMALLINT	Specifies an array of SQLUSMALLINT values that are used to ignore a parameter by the driver during execution of an SQL statement. Each value is set to either SQL_PARAM_PROCEED for the parameter to be executed or SQL_PARAM_IGNORE for the parameter to be ignored.	`SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_OPERATION_PTR, paramOperation, SQL_IS_POINTER);`

Statement Attribute	Type	Description	Example
SQL_ATTR_PARAM_STATUS_PTR	SQLUSMALLINT	Specifies an array of SQLUSMALLINT that contains the status information for each row of parameter values after calling SQLExecute or SQLExecDirect. This field is useful only if PARAMSET_SIZE is greater than 1. The parameter status for each row is set as follows: • SQL_PARAM_SUCCESS: The SQL statement was successfully executed for this set of parameters. • SQL_PARAM_SUCCESS_WITH_INFO: The SQL statement was successfully executed for this set of parameters. However, a warning has occurred. • SQL_PARAM_ERROR: An error occurred when processing this set of parameters. • SQL_PARAM_UNUSED: This parameter set was unused, possibly due to some error caused by a previous parameter.	`SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_STATUS_PTR, paramStatus, SQL_IS_POINTER);`
SQL_ATTR_PARAMS_PROCESSED_PTR	SQLULEN	Specifies a buffer in which to return the number of sets of parameters that have been processed by the driver, including error sets.	`SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMS_PROCESSED_PTR, paramsProcessed, SQL_IS_POINTER);`
SQL_ATTR_ROW_ARRAY_SIZE	SQLULEN	Specifies the number of rows returned by each call to SQLFetch or SQLFetchScroll. The default value is 1.	`SQLSetStmtAttr(pstmt, SQL_ATTR_ROW_ARRAY_SIZE, (SQLPOINTER)rowsetSize, SQL_IS_UINTEGER);`

Statement Attribute	Type	Description	Example
SQL_ATTR_ROW_BIND_OFFSET_PTR	SQLULEN	Specifies an offset added to pointers to change binding of column data. If the offset is not set to non-null, the driver dereferences the pointer, adds the dereferenced value to each of the deferred fields in the descriptor record, and uses the new pointer values when binding.	`SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_BIND_OFFSET_PTR, bindOffset, SQL_IS_POINTER);`
SQL_ATTR_ROW_BIND_TYPE	SQLULEN	Specifies the binding orientation used when SQLExtendedFetch is called. Column-wise binding is selected by setting the value SQL_BIND_BY_COLUMN. Row-wise binding is selected by setting the value to the length of a structure or an instance of a buffer into which result columns will be bound.	`SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_BIND_TYPE, (SQLPOINTER)SQL_BIND_BY_COLUMN, SQL_IS_UINTEGER);`
SQL_ATTR_ROWS_FETCHED_PTR	SQLULEN	Specifies a buffer in which to return the number of rows fetched after a call to SQLFetch or SQLFetchScroll.	`SQLSetStmtAttr(hstmt, SQL_ATTR_ROWS_FETCHED_PTR, rowsFetched, SQL_IS_POINTER);`
SQL_ATTR_ROW_OPERATION_PTR	SQLUSMALLINT	Specifies an array of SQLUSMALLINT values used to ignore a row during a bulk operation using SQLSetPos. Each value is set as follows: • SQL_ROW_PROCEED: For the row to be included in the bulk operation. • SQL_ROW_IGNORE: For the row to be excluded from the bulk operation. Rows cannot be ignored by using this array during calls to SQLBulkOperations.	`SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_OPERATION_PTR, rowOperation, SQL_IS_POINTER);`

Statement Attribute	Type	Description	Example
SQL_ATTR_ROW_STATUS_PTR	SQLUSMALLINT	Specifies an array of SQLUSMALLINT values that contain the status information of each row after a call to SQLFetch or SQLFetchScroll.	`SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_STATUS_PTR, rowStatus, SQL_IS_POINTER);`

Related Information

Collecting Server Performance Statistics [page 653]

4.5.2.1 Collecting Server Performance Statistics

Use SQLGetStmtAttr with the SQL_ATTR_SERVER_ statement attributes to retrieve server performance data with a SQLBIGINT value.

To collect memory consumption data you must enable memory tracking. In the resource_tracking section of `global.ini` set enable_tracking and memory_tracking to "on". For performance reasons, resource tracking is disabled by default.

Memory consumption data and CPU time values are 0 when memory consumption or CPU time is not available. This can occur when the above server configuration in the global.ini setting is not set to "on", or there are no requests being made to the server, or the SAP HANA server version is earlier than SAP HANA 2.0 SPS 04.

Depending on the operations performed on the statement, the resulting values can differ:

- if the statement is prepared with SQLPrepare but not executed, then the resulting values are for the prepare
- if the statement is executed with SQLExecute but not fetched, then the resulting values are for the most recent execution
- if the statement is executed with SQLExecDirect but not fetched, then the resulting values are for the ExecDirect only
- if the statement is fetched, then the resulting values are for the fetches that have occurred since the beginning of the most recent result set
- if SQLMoreResults is called, then the resulting values are reset for the new result set

> ### Note
>
> By default, execute operations (SQLExecute and SQLExecDirect) include fetch times and memory usage data for the first 32 rows. If fetching more rows requires additional fetch requests to the server, then the fetch time and memory usage value is non-zero and calculated from the beginning of the current result set.

Example

To gather memory usage data, ensure that monitoring is enabled in the resource_tracking section of `global.ini` by setting enable_tracking and memory_tracking to "on".

Prepare, Execute, and Fetch

The following SQLGetStmtAttr statements retrieve server statistics related to the prepare, execute, and fetch of the statement, including statistic summaries:

```
SQLBIGINT cpuTime = 0, processingTime = 0, memoryUsage = 0;
SQLBIGINT totalCpuTime = 0, totalProcessingTime = 0, totalMemoryUsage = 0;
SQLPrepare(stmt, (SQLCHAR*)"SELECT TOP 1000 * FROM OBJECTS", SQL_NTS);
SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_CPU_TIME, &cpuTime, SQL_IS_INTEGER, NULL);
SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_PROCESSING_TIME, &processingTime,
SQL_IS_INTEGER, NULL);
SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_MEMORY_USAGE, &memoryUsage, SQL_IS_INTEGER,
NULL);
std::cout << "PREPARE SERVER CPU: " << cpuTime << "us, PROCESSING TIME "
          << processingTime << "us, MEMORY: " << memoryUsage << " bytes\n";
totalCpuTime += cpuTime;
totalProcessingTime += processingTime;
totalMemoryUsage = memoryUsage;
SQLExecute(stmt);
SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_CPU_TIME, &cpuTime, SQL_IS_INTEGER, NULL);
SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_PROCESSING_TIME, &processingTime,
SQL_IS_INTEGER,
   NULL);
SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_MEMORY_USAGE, &memoryUsage, SQL_IS_INTEGER,
NULL);
totalCpuTime += cpuTime;
totalProcessingTime += processingTime;
if(memoryUsage > totalMemoryUsage) totalMemoryUsage = memoryUsage;
std::cout << "EXECUTE SERVER CPU: " << cpuTime << "us, PROCESSING TIME "
          << processingTime << "us, MEMORY: " << memoryUsage << " bytes\n";
SQLRETURN rc;
do {
    rc = SQLFetch(stmt);
} while(rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO);
SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_CPU_TIME, &cpuTime, SQL_IS_INTEGER, NULL);
SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_PROCESSING_TIME, &processingTime,
SQL_IS_INTEGER,
    NULL);
SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_MEMORY_USAGE, &memoryUsage, SQL_IS_INTEGER,
NULL);
totalCpuTime += cpuTime;
totalProcessingTime += processingTime;
if(memoryUsage > totalMemoryUsage) totalMemoryUsage = memoryUsage;
std::cout << "FETCH SERVER CPU: " << cpuTime << "us, PROCESSING TIME "
          << processingTime << "us, MEMORY: " << memoryUsage << " bytes\n";
std::cout << "TOTAL SERVER CPU: " << totalCpuTime << "us, PROCESSING TIME "
          << totalProcessingTime << "us, MEMORY: " << totalMemoryUsage << "
bytes\n";
```

Direct Execute

Run the following INSERT statement to retrieve server statistics related to the direct execution:

```
SQLExecDirect(stmt, (SQLCHAR*)"CREATE TABLE monitor_tab( c1 int,
       c2 int)", SQL_NTS);
    SQLCHAR insertStmt[] = "INSERT INTO monitor_tab values( 1, 1 )";
    SQLExecDirect(stmt, insertStmt, SQL_NTS);
    SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_CPU_TIME, &cpuTime,
       SQL_IS_INTEGER, NULL);
    SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_MEMORY_USAGE, &memoryUsage,
```

```
       SQL_IS_INTEGER, NULL);
    SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_PROCESSING_TIME, &processingTime,
       SQL_IS_INTEGER, NULL);
```

Direct Execute with Parameters

Run the following INSERT statement to retrieve server statistics related to the direct execution with parameters:

```
SQLExecDirect(stmt, (SQLCHAR*)"CREATE TABLE monitor_table(
      a integer primary key)", SQL_NTS);
    SQLCHAR insertStatement[] = "INSERT INTO monitor_table values(?)";
    int num_rows = 40;
    Int *data = new int[num_rows];
    SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0,
       data, sizeof(data), 0);
    for(int i=0; i<num_rows; i++) {
      data[i] = i+1;
    }
    SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)(SQLULEN) num_rows,
0);
    SQLExecDirect(stmt, insertStatement, SQL_NTS);
    SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_CPU_TIME, &cpuTime, SQL_IS_INTEGER,
       NULL);
    SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_MEMORY_USAGE, &memoryUsage,
       SQL_IS_INTEGER, NULL);
    SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_PROCESSING_TIME, &processingTime,
       SQL_IS_INTEGER, NULL);
```

Fetch

Run the following INSERT statement to retrieve server statistics related to the fetch:

```
SQLCHAR selectStmt[] = "select top 1000 * from objects";
    SQLExecDirect(stmt, selectStmt, SQL_NTS);
    SQLRETURN rc = SQLFetch(stmt);
    while(rc == SQL_SUCCESS) {
      rc = SQLFetch(stmt);
    }
    SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_CPU_TIME, &cpuTime, SQL_IS_INTEGER,
       NULL);
    SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_MEMORY_USAGE, &memoryUsage,
       SQL_IS_INTEGER, NULL);
    SQLGetStmtAttr(stmt, SQL_ATTR_SERVER_PROCESSING_TIME, &processingTime,
       SQL_IS_INTEGER, NULL);
```

Related Information

ODBC Statement Attributes [page 645]

4.5.3 ODBC Tracing and Trace Options

Activate ODBC tracing to find errors while your application is connected to a database via ODBC.

Use hdbodbc_cons to set tracing commands to initiate diagnostic tracing if you encounter either of the following issues:

- You experience problems while using an application that is based on the ODBC interface.
- You must provide an SAP HANA ODBC client trace (also known as an ODBC trace) of the application workflow that reproduces a specific problem.

> **Note**
>
> - Enabling ODBC tracing results in a significant reduction in ODBC performance.
> - Enabling tracing can result in sensitive information (such as connection properties, SQL text, and result set values) being included in the trace file.
> - If you are running your SAP HANA client on a Microsoft Windows 32-bit system, then use hdbodbc_cons32 and not hdbodbc_cons.

Trace Options

Option	Command Line Option	Description
Help	`hdbodbc_cons -h`	Displays the command line help.
Trace flush	`hdbodbc_cons [CONFIG] TRACE FLUSH ON | OFF`	Controls the use of the internal buffer (4 KB) for buffering trace content before writing to the trace file. This setting helps reduce I/O activity when tracing is enabled and improves performance.

Option	Command Line Option	Description

Trace only on an error	All on one line:
	`hdbodbc_cons [CONFIG] TRACE ONLY ON ERROR <error_code_1> [,<error_code_2>,...<error_code_1000>] [BUFFER <buffer_size>] [COUNT <count>]`
		Enables/disables content tracing if specified error codes occur a specified number of times. The maximum number of error codes that can be tracked is 1,000 and at least 1 error code must be specified. When this tracing is enabled, and a tracked error occurs, there is a `<buffer_size>`/2 bytes of tracing content before the error and a `<buffer_size>`/2 bytes of tracing content after the error are traced. `<buffer_size>` and `<count>` are optional, where the defaults are 4 KB and 1, respectively.
		In order for error content tracing to work most effectively, category tracing must also be enabled, for example:
		`hdbsqldbc_cons trace debug on level fatal`
		`hdbsqldbc_cons trace debug`

Option	Command Line Option	Description
Trace category	`hdbodbc_cons [CONFIG] TRACE <category> ON [LEVEL <severity>] | OFF`	Enables/disables a tracing category for all active and new processes. `<severity>` specifies the type of content that should be traced from one of the supported levels: • FATAL • ERROR • WARNING • INFO • DEBUG (default) Each higher level contains the information from the previous level, for example FATAL traces the least amount of content for the category while DEBUG traces all of the content for the category. The following `<category>` values are supported: **API** Traces ODBC API calls. **SQL** Traces SQL statements, results, and parameters. All `<severity>` levels apply to this category. **DEBUG** Traces the internal debug code in the client. All `<severity>` levels apply to this category. **PACKET** Traces network command protocol packets. `<severity>` does not apply to this category. **DISTRIBUTION** Traces distributed connections. This option is useful for tracking *10108 Session has been reconnected* errors. All `<severity>` levels apply to this category. **TIMING** Traces the elapsed, client, network and server time

Option	Command Line Option	Description
		for each call into SQLDBC. `<severity>` is not used with this category.
Show trace settings	`hdbodbc_cons SHOW ALL`	Displays the current trace settings.
Compact the trace	`hdbodbc_cons COMPACT`	Compacts the trace configuration store of ODBC-based clients. Use the COMPACT option on clients where trace configuration updates are performed frequently, for example, development or support systems.
Turn on a trace level for all processes	`hdbodbc_cons TRACE <level> ON`	Activates a tracing level for all active and new processes, including: **API** Traces ODBC API calls. **SQL** Traces SQL statements, results, and parameters. **DEBUG** Traces the internal debug code in the client. **PACKET** Traces network command protocol packets. **DISTRIBUTION** Traces distributed connections. This option is useful for tracking *10108 Session has been reconnected* errors. All `<severity>` levels apply to this category.

Option	Command Line Option	Description
Turn on a trace level for new processes	`hdbodbc_cons CONFIG TRACE <level> ON`	Activates a tracing level for all active and new processes, including: **SQL** Traces SQL statements, results, and parameters. **DEBUG** Traces the internal debug code in the client. **PACKET** Traces network command protocol packets. **DISTRIBUTION** Traces distributed connections. This option is useful for tracking *10108 Session has been reconnected* errors. **API** Traces driver API calls. This category is only supported for the Node.js driver. **TIMING** Traces the elapsed, client, network and server time for each call into SQLDBC.

Turn off a trace level for all processes	All processes:
	`hdbodbc_cons TRACE <level> OFF`
	New processes:
	`hdbodbc_cons CONFIG TRACE <level> OFF`
	All tracing levels:
	`hdbodbc_cons TRACE OFF`
		Deactivates a tracing level for all active and new processes.

Option	Command Line Option	Description
Set trace file name	`hdbodbc_cons CONFIG TRACE FILENAME /tmp/ODBC-%p.txt`	Sets a trace output file name. If *%p* is specified in the file name, then it is replaced with the process ID of the traced application. For clarity, and to avoid issues where only one process can write to the trace file, include *%p* in the file name so that each process writes its own trace file. The special file names "stdout" and "stderr" are supported to send trace information to stdout and stderr, respectively.
Trace encrypted content	`hdbodbc_cons [CONFIG] TRACE CSE ON|OFF`	Turns tracing on or off for content that is either sent or retrieved from a client-side encrypted column.

Remarks

You must have an SAP HANA client installed on your computer.

The hdbodbc_cons utility is installed by the SAP HANA client installation. By default, it can be found in the following locations:

On Linux and UNIX platforms	`/usr/sap/hdbclient`
On Microsoft Windows platforms	`C:\Program Files\sap\hdbclient` (64-bit client) `C:\Program Files (x86)\sap\hdbclient` (32-bit client)

ODBC trace files output the enabled trace settings at the start of every new file, even when the trace settings have changed while the SQLDBC-based application is running. The output is formatted as follows:

```
[<boolean_trace_category>
   | <trace_category_with_level>: level <level>
   | Packet Trace: [No Size Limit | <size> bytes]
   | Trace File Size Limit: <size> bytes
   | Stop On Error Code (After <count> Occurence[s]*): <error_code>
   | Trace Only On Error Codes ([Max <count> | Unlimited] Occurence[s]*, Buffer
Size = <buffer_size>): <error_code_1>[,...,<error_code_N>]* ]+
```

Example

The following example shows how to trace a connection using the ODBC tracing tool:

```
hdbodbc_cons SHOW ALL
```

```
hdbodbc_cons CONFIG TRACE FILENAME /tmp/ODBC-%p.txt
hdbodbc_cons TRACE API ON
hdbodbc_cons TRACE SQL ON LEVEL INFO
… reproduce issue …
hdbodbc_cons TRACE OFF
```

The following example shows how to use environmnet variables to trace an application using the Microsoft Windows command prompt:

```
set HDB_ODBC_TRACEFILE="%TEMP%/odbctrace.txt"
set HDB_ODBC_TRACEOPTS="DEBUG=ON,SQL=ON"
...reproduce issue using the client...
set HDB_ODBC_TRACEFILE=
set HDB_ODBC_TRACEOPTS=
```

The following example shows how to use environmnet variables to trace an application using the UNIX/Bash (or other equivalent) shell:

```
export HDB_ODBC_TRACEFILE="/tmp/odbctrace.txt"
export HDB_ODBC_TRACEOPTS="DEBUG=ON,SQL=ON"
...reproduce issue using the client...
unset HDB_ODBC_TRACEFILE
unset HDB_ODBC_TRACEOPTS
```

Related Information

Troubleshooting: ODBC Tracing [page 662]
Troubleshooting: SQLDBC Tracing [page 794]
SQLDBC Tracing [page 785]

4.5.3.1 Troubleshooting: ODBC Tracing

Identify issues affecting your ODBC trace.

If no trace file is written to the disk, then verify whether any of the following issues are affecting your trace session:

- Ensure that you are tracing an ODBC-based application as hdbodbc_cons only applies to ODBC. To trace a SQLDBC-based application, use hdbsqldbc_cons.
- Ensure that you have enabled tracing on the correct machine. If several application servers are being used in a landscape, then determine which application server the application is running on.
- Ensure that the user running hdbodbc_cons is the same user running the application being traced.
- Ensure that the trace tool has the same version as the driver. The hdbodbc_cons version should correspond to `libodbcHDB.so` / `libodbcHDB.dll`.
- Ensure that you specified a trace file location that is accessible from the application and in which the user is allowed to write files.
- Ensure that you look for the trace file in the correct output directory. The trace file name is relative, so the trace is written in the working directory of the application. If necessary, specify an absolute path.

- Ensure that you include *%p* in the trace file name. If you did not specify *%p* and more than one process is using the interface, then only the first process can write into the trace file and you receive a trace file that may not contain the correct information. The trace file path might be overloaded by ABAP profile parameters, in which case the trace files are located in the instance's work directory.

If you are working on Microsoft Windows, then verify whether any of the following issues are affecting your trace session:

- Ensure whether or not you need to use the -u `<user>` option to specify the SAP service `<SID>` username for the process.
- Ensure whether or not the user running hdbodbc_cons with -u has administrator privileges. To change trace settings for other users, administrator privileges are required. Run hdbodbc_cons from a new command prompt using the *Run as administrator* option.

Related Information

Troubleshooting: SQLDBC Tracing [page 794]
ODBC Tracing and Trace Options [page 656]
SQLDBC Tracing [page 785]

4.5.3.2 Callback for ODBC Connection-level Tracing

ODBC applications provide a callback to receive trace messages for a particular connection.

Use the following prototype to implement a callback:

```
typedef void (*ODBC_TraceCallback) (const char *category, int level, const char
*message, size_t size, void *context);
```

A callback is provided by setting the following connection attributes:

- `SQL_ATTR_TRACE_CALLBACK`
- (Optional) `SQL_ATTR_TRACE_CALLBACK_CONTEXT`
- `SQL_ATTR_TRACE_CALLBACK_OPTIONS`

The context parameter provided by the attribute SQL_ATTR_TRACE_CALLBACK_CONTEXT goes back as the context parameter to the trace callback by the ODBC driver.

> **Note**
>
> `SQLSetConnectAttr()` is called separately for each attribute.

The client calls the callback once for each trace message and blocks until the callback returns.

The category parameter passed to the callback is a string representing the trace category of the message. The `ODBC_TRACE_LEVEL` parameter is an integer representing the severity level of the message with the following pre-defined values:

```
enum ODBC_TraceLevel {
```

```
    ODBC_TRACE_LEVEL_NONE = 0x0,
    ODBC_TRACE_LEVEL_FATAL = 0x1,
    ODBC_TRACE_LEVEL_ERROR = 0x2,
    ODBC_TRACE_LEVEL_WARNING = 0x3,
    ODBC_TRACE_LEVEL_INFO = 0x4,
    ODBC_TRACE_LEVEL_DEBUG = 0xF
};
```

Enable Multiple Trace Categories

Calling `SQLSetConnectAttr()` multiple times for an attribute replaces the previous value with the new value. For example, passing DEBUG for the options in one call and then API in a second call will not turn on both DEBUG and API tracing, but will disable DEBUG tracing and enable only API tracing. To enable multiple trace categories, provide all of them in a single trace option string:

```
SQLSetConnectAttr(hdbc, SQL_ATTR_TRACE_CALLBACK_OPTIONS, "DEBUG,API,SQL",
SQL_NTS);
```

Example

The following example shows how to set up a connection trace callback:

```
static void MyDBConnection::traceCallback(const char *category, int level, const
char *message, size_t size, void *context)
{
    static_cast<MyDBConnection*>(context)->log(message, size);
}
void MyDBConnection::connect()
{
    SQLAllocHandle(SQL_HANDLE_DBC, m_henv, &m_hdbc);
    const char *traceOptions = "DEBUG,API,SQL";
    SQLSetConnectAttr(m_hdbc, SQL_ATTR_TRACE_CALLBACK,
(SQLPOINTER)traceCallback, 0);
    SQLSetConnectAttr(m_hdbc, SQL_ATTR_TRACE_CALLBACK_CONTEXT, (SQLPOINTER)this,
0);
    SQLSetConnectAttr(m_hdbc, SQL_ATTR_TRACE_CALLBACK_OPTIONS,
(SQLPOINTER)traceOptions, SQL_NTS);
    SQLDriverConnect(m_hdbc, ...)
}
```

Related Information

ODBC Connection Attributes [page 614]
SQLDBC Trace Options [page 786]

4.6 Python Application Programming

The Python Database API Specification defines a set of methods that provides a consistent database interface independent of the actual database being used.

The Python extension module for SAP HANA implements PEP 249. Once you install the module, you can access and change the information in databases from Python. In PEP 249, autocommit is turned off by default. In the SAP HANA Python driver, autocommit is turned on by default.

For information, see PEP 249 -- Python Database API Specification v2.0 .

For a list of supported Python versions, see 3165810

Related Information

SAP Note 3136015

4.6.1 Install the Python Driver

Install the Python driver into your local Python environment by using the pip installer so that you can use the Python driver to connect to an SAP HANA database.

Prerequisites

You have run the SAP HANA client install.

Procedure

1. If you have multiple Python environments, then activate the Python environment where you want to install the Python driver. If you don't have multiple Python environments, then ignore this step.
2. Run the install:

Platform	Command

UNIX, Linux, and macOS	Use one of the following install methods:
	• pip install
	`pip install /usr/sap/hdbclient/hdbcli-N.N.N.tar.gz`
	• easy_install
	`easy_install /usr/sap/hdbclient/hdbcli-N.N.N.tar.gz`
	• disutils installer
	`tar -xf hdbcli-N.N.N.tar.gz`
	Then run:
	`cd hdbcli-N.N.N`
	Lastly, run:
	`setup.py install`
	• To install the Python driver only for the current user, then add `--user` after "install" to any of the above commands.

Microsoft Windows	Use one of the following install methods:
	• pip install
	`pip install "C:\Program Files\SAP\hdbclient\hdbcli-N.N.N.zip"`
	• easy_install
	`easy_install "C:\Program Files\SAP\hdbclient\hdbcli-N.N.N.zip"`
	• disutils installer
	Unzip `hdbcli-N.N.N.zip` into a directory, then run:
	`cd hdbcli-N.N.N`
	Lastly, run:
	`setup.py install`
	• To install the Python driver only for the current user, then add `--user` after "install" to any of the above commands.

Platform	Command

Python Package Index	Use the following install methods:
	• PyPI
	`pip install hdbcli`
	• pip install (preferred)
	`python -m pip install hdbcli==N.N.N`
	• easy_install (deprecated)
	`-m easy_install hdbcli==N.N.N`
	• To install the Python driver only for the current user, then add `--user` after "install" to any of the above commands.

Related Information

SAP Note 3136015

4.6.2 Connect to SAP HANA from Python

Use the connect method to connect to an SAP HANA database.

Context

The Python driver supports all of the SQLDBC connection properties.

You can trace Python connections by using SQLDBC tracing.

In PEP 249, autocommit is turned off by default. In the SAP HANA Python driver, autocommit is turned on by default.

Procedure

1. Install the Python driver.
2. Import the dbapi module:

```
from hdbcli import dbapi
```

3. Use the connect method:

   SAP HANA Platform:

   SAP HANA Cloud:

```
dbapi.connect(address=<hostname>, port=443, user='system',
password='Password123')
```

Related Information

Use the User Store (hdbuserstore) [page 856]
SQLDBC Tracing [page 785]

4.6.3 Uninstall the Python Driver

Remove the files for the Python driver from your computer.

Context

These instructions are for uninstalling the Python driver by using pip.

Procedure

Run the following command.

```
$ pip uninstall hdbcli
```

4.6.4 Connect Method and Python Connection Properties

Creates a connection to an SAP HANA database.

Syntax

```
dbapi.connect( <connection-property>[, ... ] )
```

Remarks

All of the connection properties are optional and case insensitive unless otherwise noted. Specify connection parameters as key-value pairs in a comma-delimited list.

> **Note**
>
> Do not set connection properties to None or an empty string. The client will substitute any None and empty string values with the property's default setting.

Connection Properties Specific to the Python Driver

Property	Value	Default Value	Description
address	`<string>`	Empty string	Specifies the IP address or host name of the database instance. This property is case sensitive.
autocommit	boolean	True	Sets the autocommit mode for the connection. In autocommit mode, every statement is automatically committed. Otherwise, commits and/or rollbacks must be done manually. This property is case sensitive.
packetSize	integer	None	Sets the communication packet size for the connection.
password	`<string>`	Empty string	Specifies the password for the user. This property is case sensitive.
port	integer	0	Specifies the port number of the database instance.
properties	`<string>`		additional dict object with special properties

Property	Value	Default Value	Description
user	`<string>`	Empty string	Specifies the user name. This property is case sensitive.

SQLDBC Connection Properties

The following table lists SQLDBC connection properties (property names are case insensitive) that can also be used with Python:

> **Note**
>
> A valid setting for a Boolean property can be: “1”, “TRUE”, “YES”, or “ON” for a TRUE setting or “0”, “FALSE”, “NO”, or “OFF” for a FALSE setting. The setting of a Boolean property is case-insensitive.

Property	Value	Default	Description
`abapVarCharMode`	Boolean	FALSE	Specifies that in ABAP VARCHAR mode, a single space and an empty string are considered the same.
`allowLocalCompress`	Boolean	TRUE	Determines whether or not connections from the same machine as the SAP HANA server are compressed. The `compress` parameter and/or the `indexserver.ini` > *session* > *compression* server configuration control whether or not network compression is enabled. **Note** If the connection uses a proxy, do not use `allowLocalCompress`=FALSE as the connection may not be compressed when required.

Property	Value	Default	Description
`allowReconnectOnSelect`	Boolean	OFF	Allows reconnects to happen for SELECT queries which fail during a receive. By default, this property is set to OFF, which allows for backward compatibility.
`bindAddress`	`<ip-address>`	NULL	Specifies that the provided address is used as the source address for the socket connection. By default, the address preferred by the OS is used. If setting `bindAddress` to a hostname, that hostname should only resolve to a single IP address. In the majority of network configurations, specifying `bindAddress` is not required. **Note** Only specify `bindAddress` if it is required to connect in your network configuration.
`chopBlanks`	Boolean	FALSE	Removes trailing blanks from character output.
`chopBlanksInput`	Boolean	FALSE	Removes trailing blanks from character input.

Property	Value	Default	Description
`communicationTimeout`	Timeout in milliseconds	0 (disabled)	Aborts communication attempts to the server after the specified timeout. If set to a non-zero value, the driver will wait for a maximum of the specified number of milliseconds for any request to the server to complete. If the time has expired and a response has not been received, the driver returns an error.
`compress`	Boolean	FALSE	Enables or disables network compression for the connection. The `indexserver.ini` > *session* > *compression* server configuration defaults to FALSE. Setting this parameter overrides the setting of the `indexserver.ini` > *session* > *compression* parameter. The `allowLocalCompress` property controls if clients running on the same machine as the server use network compression when compression is requested.

Property	Value	Default	Description
`connDownRollbackError`	Boolean	FALSE	Setting this property to TRUE specifies that if the database connection is down due to network or SAP HANA server problems, the client rolls back the last transaction and sends and error to the application. The application can resend the last transaction to the server after properly handling the error. The following conditions must be true: • The session is recoverable in the server • There is a written transaction (uncommitted operation) in the session • The client failed to send the last SQL statement, or the last statement was sent to the server successfully, but both auto-commit and DDL auto-commit in the client are off. By default, this property is set to FALSE.
`connectTimeout`	Timeout in milliseconds	0 (limited by the `nodeConnectTimeout` connection property)	Aborts connection attempts after the specified timeout. `connectTimeout` limits the overall time for the entire connection to try all server addresses specified. See `nodeConnectTimeout` for per-node timeout.
`currentSchema`	`<schema>`	CURRENTUSER	Sets the current schema, which is used for identifiers without a schema.

Property	Value	Default	Description
`cursorHoldabilityType`	NONE, COMMIT, ROLLBACK, COMMIT_AND_ROLLBACK	COMMIT	Specifies whether the cursor for a result set remains open after the transaction is committed or rolled back. If a cursor remains open, then the application is able to fetch more rows from it. NONE means the cursor is always closed after a commit or rollback, while COMMIT means the cursor remains valid after a commit, but not after rollback. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources. Support for holding a cursor over ROLLBACK or COMMIT_AND_ROLLBACK is available in SAP HANA servers beginning with SAP HANA 2.4.

Property	Value	Default	Description
`emptyTimestampIsNull`	Boolean	TRUE	When enabled, DATE, TIME, SECONDDATE, and TIMESTAMP values inserted as empty strings are returned as NULLs. When disabled, these values are returned as the following out-of-band values: • `DATE` - one day before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 00:00.000000000 UTC BC • `TIME` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `SECONDDATE` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `TIMESTAMP` - one hundred nanoseconds before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.999999900 UTC BC

Property	Value	Default	Description
`ignoreTopology`	Boolean	FALSE	Controls how the client uses the topology returned from the server. TRUE indicates to use the topology unless port-forwarding is detected and FALSE indicates to always ignore the topology. It is an error to specify both ignoreTopology=TRUE and siteType=PRIMARY or SECONDARY.
`isolationLevel`	TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE	TRANSACTION_READ_COMMITTED	Sets the isolation level for the connection.
`isoTimestampOutput`	`<string>`	FALSE	Displays whether or not 'T' is used as the separator between the Date and Time parts of a timestamp retrieved as a string: TRUE/FALSE. If FALSE, then a space character is used as the separator.
`key`	`<username>`	Empty	Specifies the key for the hdbuserstore.
`maxLazyDroppedStatements`	`<number>`	50	Controls the maximum number of queued statements to be dropped. Setting this value to 0 disables the lazy drop feature and causes a request to be sent to the server each time a statement is dropped by the application.
`maxVersion`	`<string>`		Specifies the maximum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.

Property	Value	Default	Description
`minVersion`	`<string>`		Specifies the minimum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.
`nodeConnectTimeout`	Timeout in milliseconds	60000	Aborts the connection attempt after the specified maximum timeout. The connection attempt typically fails immediately if the server cannot be found. If connection attempts to a particular server address are slow to fail, then using this connection property reduces the time before a successful connection by minimizing the time waiting to connect to individual addresses. On a host with multiple resolved IP addresses, the network timeout for each attempt to connect to an IP address will be equal to the `nodeConnectTimeout` divided by the number of resolved IP addresses. If this option is set to zero, the `connectTimeout` setting will be used. If both of the `nodeConnectTimeout` and `connectTimeout` are set to zero, then SQLDBC will use the system TCP/IP socket connect timeout to connect to the IP address.

Property	Value	Default	Description
`packetCaching`	Boolean	TRUE	Caching packets reduces the need to continually allocate and deallocate memory for packets. However, this increases the peak memory usage of the application, and for how long it stays at or near that peak.
`packetSize`	`<number>` in bytes	1048576 (1 MB)	Sets the maximum size of a request packet sent from the client to the server, in bytes. In some cases, request packets can be larger (for example, if a SQL statement or parameter value is larger than the packetSize). The minimum is 1 MB.
`packetSizeLimit`	`<number>` in bytes	2147483647 (2 GB)	Sets the maximum size for a request or reply packet, in bytes. If SQLDBC needs to allocate a packet larger than the limit, then it returns an error. If a reply packet is too large, then the connection is dropped. The minimum value is the packetSize value; the maximum value is 2 GB.

Property	Value	Default	Description
`prefetch`	Boolean	TRUE	Controls result set prefetching. When enabled, result set prefetch requests the next block of rows of a result set while the application is still processing the current block of rows. When fetching large result sets, result set prefetch can improve performance; however, it can also result in the client using more memory. In some cases, the server JobExecutor thread could be blocked from sending the reply for an indefinite period of time until the client receives the prefetch reply. If the result set contains LOB columns, then result set prefetch is always disabled.
`{ proxyHostname | proxy_host }`	`<host-name>`		Specifies the host name of the proxy server. The default proxy server is SOCKS5. If `proxyHttp` is set to true, the proxy server is assumed to be an HTTP proxy.
`proxyHttp`	Boolean	FALSE	When set to TRUE, enables HTTP proxy authentication and implies that ENCRYPT=TRUE. This property is only applicable if `proxyHostname` is specified and `webSocketURL` is not specified.

Property	Value	Default	Description
`{ proxyPassword | proxy_password }`	`<password>`		Specifies the password for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the Cloud Connector location ID value. The value must be entered as a base64 encoded value with a "1." prepended to the string.
`{ proxyPort | proxy_port }`	`<port-number>`		Specifies the port of the proxy server. The default is 1080.
`{ proxyScpAccount | proxy_scp_account }`	SAP BTP, Cloud Foundry Environment: `<location-ID>` SAP BTP, Neo Environment: `<subaccount>.<location-ID>`		Provides SAP Cloud Connector routing information to the Cloud Connector SOCKS proxy. In an SAP BTP, Cloud Foundry Environment, provide an optional `<locationID>` in plaintext. Internally, the value is converted to base64 encoding. This value overrides the `proxyPassword` connection property. In an SAP BTP, Neo Environment, either `proxyScpAccount` or `proxyUsername` must be specified. `<subaccount>` is mandatory and `<locationID>` is optional. Both values are entered as plaintext strings and are internally converted to base64 encoding. This value overrides the `proxyUsername` connection property.

Property	Value	Default	Description
`{ proxyUserName | proxy_userid }`	`<user-name>`		Specifies the user name for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the JSON Web Token (JWT) value. When using an SAP HANA Cloud Connector in the SAP BTP, Neo Environment, this field is used to provide the subaccount and location ID details. The values must be base64 encoded manually using the format `1.base64(subaccount).base64(locationID)`.
`reconnect`	Boolean	TRUE	Controls whether the system automatically reconnects to the database instance after a command timeout or when the connection is broken. Reconnecting restores the old state (for example, if no transaction was open).

Property	Value	Default	Description
`resolveHostName`	IPv4First IPv6First	IPv4First	Controls which addresses to use in a server address, especially if an address or hostname resolves to multiple IP addresses. In all cases, stop attempting further resolved addresses once a TCP/IP connection is successfully established. The following values can be used with `resolveHostName`: • IPv4First (default) - try all resolved addresses starting with IPv4 addresses. • IPv6First - try all resolved addresses starting with IPv6 addresses.
`sessionVariable:<key-name>`	`<session variable value>`	None except for APPLICATION and APPLICATIONUSER keys	Sets the session variable `<key-name>` to `<session variable value>`. For example, to set the session variable with key `APPLICATION`, use `sessionVariable:APPLICATION=MyApplication`.

Property	Value	Default	Description
`statementCacheSize`	`<number>`	0 (no statements cached)	Controls the maximum number of cached prepared statements. Caching SQL statements that are frequently prepared and dropped can improve performance. DDL can cause behavior changes when statement caching is enabled. **Note** If it is possible that the definition of objects referred to by the prepared statements may change because DDL is executed while you are connected, then it is recommended that you do not enable prepared statement caching.
`statementRoutingFailureBackoff`	Boolean	TRUE	Enables or disables the re-establishment of an internal connection to a specific host for a time period, after the internal connection fails during the preparation or execution of a statement routed to that host. The minimum time period is 1 minute. The time period is doubled after each subsequent internal connection establishment or usage failure for the same node, up to a maximum of 1 hour. Every successful internal connection re-establishment and usage resets the time period to the minimum value.

Property	Value	Default	Description
`statementRoutingWarnings`	Boolean	FALSE	Returns a SQL warning to the application with SQLCODE -11001 or -11003 that gives information about the routing failure of a prepared statement.
`tcpKeepAliveCount`	`<number-of-retries>` This value is ignored if it is 0.	5	Controls the number of KeepAlive packets sent to the server before the driver assumes that the connection is closed. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.
`tcpKeepAliveIdle`	`<seconds>` This value is ignored if it is 0.	200	Controls the number of seconds that the connection is idle before the driver sends a KeepAlive packet. If 0 is specified, this value is determined by the operating system.
`tcpKeepAliveInterval`	`<seconds>` This value is ignored if it is 0.	4	Controls the number of specified seconds before the driver sends an additional KeepAlive packet if there was no response to the previous KeepAlive packet. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.

Property	Value	Default	Description
`timestampPadding`	Boolean	TRUE	In a timestamp where the format contains a fraction, this property determines whether the fractional portion of the timestamp will be 7 or 9 digits. When disabled, the fractional portion is 7 digits to match how the server displays timestamps. Otherwise, it is 9 digits. **Note** ABAP timestamps are not affected regardless of the value of `timestampPadding`.
`traceFile`			Specifies the trace file name. Only the stdout and stderr values are supported.
`traceOptions`			Implicitly enables tracing for all SQLDBC-based client connections. See the HDB_SQLDBC_TRACEOPTS description in the **Environment Variables** topic for more information.
`webSocketURL` (SAP HANA Service only)	`<webSocketUrl>`		Specifies the WebSocket URL used to connect to a SAP HANA server. The connection property value cannot be empty. WebSockets connections have the following default connection property settings: • encrypt=TRUE • compress=TRUE • reconnect=FALSE

Security Connection Properties

Property	Value	Default	Description
`{ cseKeyStorePassword | clientside_encryption_keystore_password }`	`<string>`		Provides the password for the local key store. This option is required when using client-side encryption. Passwords must be at least eight characters and have at least one upper case character, one lower case character, and one number.
`authenticationMethods`	`<comma-separated-string>`		Provides a comma-separated list of authentication methods to be used by the client. Possible values are: password, pbkdf2, ldap, saml, jwt, saplogon, kerberos, sessioncookie, and X509.
`authenticationX509`	`<string>`	`<value-of-sslKeyStore>`	Provides a string containing the X509 certificate or the name of a file containing the X509 certificate. The certificate must be PEM-encoded in PKCS8 format. Defaults to the existing value of `sslKeyStore`, if specified.
`authenticationX509Password`	`<password>`		With CommonCryptoLib, specifies the PIN for the PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`encrypt`	Boolean	FALSE	Enables or disables TLS encryption. For SAP HANA versions 2.5 and lower, set encrypt=TRUE when connecting to port 443 (SAP HANA Cloud). For SAP HANA versions 2.6 and higher, encrypt=TRUE is not required when connecting to port 443.

Property	Value	Default	Description
`sslCryptoProvider`	{ commoncrypto | openssl | mscrypto }	1. commoncrypto (if installed) 2. openssl/mscrypto	Specifies the cryptographic library provider used for TLS communication. If you specify a value for this property, then you must also explicitly specify paths in both the `sslKeyStore` and `sslTrustStore` properties to avoid configuration issues. If CommonCryptoLib is not available, OpenSSL is used by default in Linux environments, and msCrypto in Microsoft Windows environments. OpenSSL is not available on Microsoft Windows. **Note** Check the `client` folder of the installation package to see if `COMMONCRYPTOLIB.TGZ` is present. If not, you can download CommonCryptoLib separately. For instructions on how to download CommonCryptoLib, see "Download and Install SAP Common Crypto Library" in the *SAP HANA Client Installation and Update Guide*.
`sslHostNameInCertificate`	`<string>`	Empty	Specifies the host name used to verify server's identity. The host name specified here verifies the identity of the server instead of the host name with which the connection was established. For example, in a single-host system, if a connection is established from a client on the same host as the server, then a mismatch would arise between the host named in the certificate (actual host name) and the host used to establish the connection (localhost). If you specify `*` as the host name, then the server's host name is not validated. Other wildcards are not permitted.

Property	Value	Default	Description
`sslKeyStore`	`<file>` | `<PEM-encoded-identity>` msCrypto: MY | Root | Trust | CA	`$SECUDIR/sapcli.pse` (CommonCryptoLib) `$HOME/.ssl/key.pem` (OpenSSL) MY (msCrypto)	Specifies the path to the keystore file that contains the client's identity. An identity is used for mutual TLS authentication and consists of the client's private key, the client's certificate, and, optionally, the certificate of the signing authority that signed the client's certificate. If you are using CommonCryptoLib, then the PSE file must also contain the trust store (for example, the server's public certificates) and the sslTrustStore property should be empty. If you are using CommonCryptoLib, but not doing mutual TLS authentication, then the PSE file contains only the trust store. If you are using CommonCryptoLib, then use the SAPGENPSE tool (installed with CommonCryptoLib) to create the client PSE (`sapcli.pse`) and generate the client's keys. You must also import the server's public certificates into `sapcli.pse`. Typically, this is the root certificate or the certificate of the certification authority that signed the server's public certificates. See *SAP Note 1718944 (SAP HANA DB: Securing External SQL Communication (CommonCryptoLib))*. If you are using OpenSSL, use OpenSSL tools to create the required keystore file. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.

Property	Value	Default	Description
`sslMinProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	TLS12	Controls the minimum TLS version. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The lowest protocol version is accepted in TLS handshakes.
`sslMaxProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	MAX	Controls the maximum TLS version. MAX allows this parameter to be future-proof in the event that new TLS versions are supported. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The highest negotiated protocol version is used in TLS handshakes.
`sslKeyStorePassword`	`<password>`		With CommonCryptoLib, specifies the PIN for the keystore PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`sslSNIHostname`	`<string>`	Empty	Specifies the name of the host that is attempting to connect at the start of the TLS handshaking process.
`sslSNIRequest`	Boolean	TRUE	Specifies whether SNI requests are enabled for TLS connections: TRUE/FALSE.

Property	Value	Default	Description
`sslTrustStore`	`<file>` | `<PEM-encoded-trust-store>` msCrypto: MY | Root | Trust | CA	`$HOME/.ssl/trust.pem` (OpenSSL) sapcli.pse (CommonCryptoLib) MY (msCrypto)	Specifies the path to a trust store file that contains the server's public certificates if using OpenSSL. Typically, the trust store contains the root certificate or the certificate of the certification authority that signed the server's public certificates. If you are using the cryptographic library CommonCryptoLib or msCrypto, leave this property empty. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.
`sslUseDefaultTrustStore`	Boolean	TRUE	(Only applies to the OpenSSL Crypto Provider) Set to FALSE in order to disable the use of the default Certificate Authorities provided with OpenSSL or the Operating System.
`sslValidateCertificate`	Boolean	TRUE	Specifies whether to validate the server's certificate.

Returns

Calling the connection method returns an instance of the connection class.

Example

Import the dbapi module and then use the connect method to connect to the on-premise SAP HANA database server:

```
from hdbcli import dbapi
connect(compress='FALSE', address='localhost', port=30015)
```

Connect by using the database parameters stored in hdbuserstore:

```
connect(key=<hdbuserstore-key>)
```

Connect to an on-premise tenant database in an MDC system:

```
address=<host>,port=30013,databasename='TENANT1'
```

4.6.5 Connection Class

Represents a connection to a database.

Remarks

The Python driver supports all SQLDBC connection properties.

You can trace Python connections by using SQLDBC tracing.

Documentation is provided for methods that are not part of PEP 249 or whose behavior differs from PEP 249.

Methods

Method	Description	Differences from PEP 249
cancel() Method [page 692]	Cancels the database request that is currently being executed on the connection.	Not defined in PEP 249.
close() Method [page 693]	Closes the connection.	No implicit rollback is performed unless autocommit is turned off.
commit() Method [page 693]	Commits any pending transactions to the database.	No implicit rollback is performed unless autocommit is turned off.
cursor() Method [page 694]	Returns a new Cursor Object using the connection.	None.
getaddress() Method [page 694]	Returns the host name that was used to connect to the database.	Not defined in PEP 249.
getautocommit() Method [page 695]	Gets the connection's autocommit mode.	Not defined in PEP 249.
getclientinfo([key]) Method [page 695]	Gets the connection's client information.	Not defined in PEP 249.
isconnected() Method [page 696]	Returns a value that indicates whether there is a connection to the database.	Not defined in PEP 249.

Method	Description	Differences from PEP 249
rollback() Method [page 696]	Performs a rollback on the connection.	Autocommit is turned on by default.
setautocommit([auto]) Method [page 697]	Sets the connection's autocommit mode.	Not defined in PEP 249.
setclientinfo(string) Method [page 698]	Sets the connection's client information.	Not defined in PEP 249.

Related Information

PEP 249 Python Database API: Connection Objects

4.6.5.1 cancel() Method

Cancels the database request that is currently being executed on the connection.

Syntax

```
connection.cancel()
```

Returns

Returns True if cancel was successful; otherwise, returns False.

Remarks

If the connection is closed, then this method throws a connection closed error.

4.6.5.2 close() Method

Closes the connection immediately.

Syntax

```
connection.close()
```

Remarks

This method never fails. You cannot use the connection once you call the `close` method. An error is raised for any operations that attempt to use the connection.

4.6.5.3 commit() Method

Commits any pending transactions to the database.

Syntax

```
connection.commit()
```

Returns

Returns None; otherwise, it throws an exception.

Remarks

By default, autocommit is turned on. Change the autocommit setting with the setautocommit method.

4.6.5.4 cursor() Method

Returns a new cursor object using the connection.

Syntax

```
connection.cursor()
```

Returns

A new cursor object using the connection.

Related Information

Cursor Class [page 698]

4.6.5.5 getaddress() Method

Returns the host name that was used to connect to the database.

Syntax

```
connection.getaddress()
```

Returns

Returns the host name that was used to connect to the database. The port number is not included.

4.6.5.6 getautocommit() Method

Gets the connection's autocommit mode.

Syntax

```
connection.getautocommit()
```

Remarks

By default, autocommit is turned on. Change the autocommit setting with the setautocommit method.

Related Information

setautocommit([auto]) Method [page 697]

4.6.5.7 getclientinfo([key]) Method

Returns the connection's client information.

Syntax

```
connection.getclientinfo([<key>])
```

Parameters

Recommended Type	Name	Description
String	<key>	Specify a supported session variable.

Returns

When <key> is specified, the connection information for the client is returned. If <key> is not specified, then a dictionary of all the clientinfo values is returned.

Related Information

Setting Session-Specific Client Information [page 16]

4.6.5.8 isconnected() Method

Returns a value that indicates whether there is a connection to the database.

Syntax

```
connection.isconnected()
```

Returns

Returns True or False.

4.6.5.9 rollback() Method

Performs a rollback on the connection.

Syntax

```
connection.rollback()
```

Remarks

By default, changes are committed before the rollback is performed. Change the autocommit setting with the setautocommit method.

Related Information

setautocommit([auto]) Method [page 697]

4.6.5.10 setautocommit([auto]) Method

Sets the connection's autocommit mode.

Syntax

```
connection.setautocommit([auto])
```

Parameters

Recommended Type	Name	Description
Boolean	`auto`	This optional parameter controls the autocommit setting for the connection. If you omit the parameter or specify True, then autocommit is enabled.

Remarks

By default, autocommit is turned on.

Related Information

getautocommit() Method [page 695]

4.6.5.11 setclientinfo(key[, value]) Method

Sets the connection's client information.

Syntax

```
connection.setclientinfo(<key>[, <value>])
```

Parameters

Recommended Type	Name	Description
String	`<key>`	Only plain, 7-bit ASCII characters are supported.
String	`<value>`	A string of Unicode characters. If you do not supply this parameter, then it clears the key (equivalent to passing value=None).

Related Information

Setting Session-Specific Client Information [page 16]
getclientinfo([key]) Method [page 695]

4.6.6 Cursor Class

Represents a database cursor. Cursors manage the context of fetch operations.

Remarks

When a connection has multiple cursors, changes made to the database by one cursor are immediately visible to other cursors. You can configure whether cursors that are created from different connections are isolated.

Documentation is provided for methods that are not part of PEP 249 or whose behavior differs from PEP 249.

Methods

Method	Description	Differences from PEP 249
callproc(procname[, parameters][, overview]) Method [page 701]	Calls the specified stored procedure from the database.	The `overview` parameter is not part of PEP 249.
close() Method [page 702]	Closes open result sets and LOBs on the cursor.	The cursor objects are reusable after calling close().
description_ext() Method [page 703]	Retrieves a sequence of result column descriptions.	Not defined in PEP 249.
execute(operation[, parameters]) Method [page 703]	Prepares and executes a database operation.	The parameters can be specified by using keywords.
executemany(operation, list_of_parameters[, batcherrors=False]) [page 704]	Prepares and executes a database operation for multiple sets of parameters. **Note** When `batcherrors` (Boolean) is TRUE: • getrowsaffectedcounts() returns a tuple of rows affected counts for all rows that were processed by the call. • If a non-python method call error occurs, executemany() raises a dbapi.ExecuteManyError exception. dbapi.ExecuteManyError is a dbapi.Error that contains a tuple of dbapi.ExecuteManyErrorEntry errors, with each having error code and text on each client/server error that occurred during the call.	The `batcherrors` parameter is not part of PEP 249.
executemanyprepared(list_of_parameters) Method [page 706]	Executes a previously prepared database operation in a batch format.	Not defined in PEP 249.
executeprepared([parameters]) Method [page 707]	Executes a previously prepared database operation.	Not defined in PEP 249.
fetchone([uselob]) Method [page 710]	Fetches the next row of a query result set.	The `uselob` parameter is not part of PEP 249.
fetchall() Method [page 708]	Fetches all of the remaining rows of a query result as an array of sequences.	None.
fetchmany([size=cursor.arraysize]) Method [page 709]	Fetches the next set of rows from a query result as an array of sequences.	None.

Method	Description	Differences from PEP 249
getpacketsize() Method [page 712]	Returns the network packet size used by the cursor.	Not defined in PEP 249.
get_resultset_holdability() Method [page 712]	Retrieves the result set holdability.	Not defined in PEP 249.
getrowsaffectedcounts() Method [page 713]	Returns a tuple containing the rows affected updates for each row that was processed by the executemany() method.	Not defined in PEP 249.
getwarning() Method [page 715]	Retrieves a warning object if there is a warning from the previous execution.	Not defined in PEP 249.
haswarning() Method [page 716]	Checks whether there are warnings from the previous execution.	Not defined in PEP 249.
nextset() Method [page 716]	Makes the cursor skip to the next result set and closes the current result set.	None.
parameter_description() Method [page 717]	Retrieves a sequence of descriptions of parameters.	Not defined in PEP 249.
prepare(operation[, newcursor=False]) Method [page 718]	Prepares a database operation.	Not defined in PEP 249.
scroll(value[, mode]) Method [page 719]	Scrolls through a result set.	Not defined in PEP 249.
server_cpu_time() Method [page 720]	Returns the total CPU processing time, in microseconds.	Not defined in PEP 249.
server_memory_usage() Method [page 721]	Returns the memory usage data on the server side, in bytes.	Not defined in PEP 249.
server_processing_time() Method [page 722]	Returns the elapsed processing time, in microseconds, on the server side.	Not defined in PEP 249.
setinputsizes(sizes) Method	Not supported.	Not supported.
setcommandinfo(command-info, line-number) Method [page 724]	Sets an arbitrary string and integer as command information to be sent to the database along with the next command, `execute`.	
setfetchsize(value) Method [page 725]	Sets the size of prefetched result sets.	Not defined in PEP 249.
setoutputsizes() Method	Not supported.	Not supported.
setpacketsize(value) Method [page 725]	Sets the network packet size used by the cursor.	Not defined in PEP 249.
set_resultset_holdability(holdability) Method [page 727]	Sets the result set holdability.	Not defined in PEP 249.

Cursor Attributes

Members	Description	Differences from PEP 249
arraysize	Specifies the cursor fetch size (number of rows). The default value is 32.	Not defined in PEP 249.
description	For operations that return rows or cursors that have invoked an operation by using the execute*() method, this attribute returns information about the rows in the result set.	None.
maxage	Specifies the refresh age for the cached view.	Not defined in PEP 249.
refreshts	Specifies the refresh timestamp for a cached view.	Not defined in PEP 249.
rowcount	Specifies the number of rows returned by the most recent execute*() method..	None.

Related Information

PEP 249 Python Database API: Cursor Objects
Packet Size [page 821]

4.6.6.1 callproc(procname[, parameters][, overview]) Method

Calls the specified stored procedure from the database.

Syntax

```
cursor.callproc(<procname>[, <parameters>][, overview])
```

Parameters

Recommended Type	Name	Description
String	`<procname>`	The name of the procedure being called.
Tuple	`<parameters>`	The parameters for the specified procedure.
Boolean	`overview`	This parameter is not defined in PEP 249.

Returns

Returns a tuple of all of the parameters. Input values are returned unchanged, while output and input/output parameters may be replaced with new values. If the procedure provides a result set as output, then make the result available through the standard fetch methods.

4.6.6.2 close() Method

Closes open result sets and LOBs on the cursor.

Syntax

```
cursor.close()
```

Returns

Always returns None.

Remarks

The cursor is reusable after calling close().

4.6.6.3 description_ext() Method

Retrieves a sequence of result column descriptions.

Syntax

```
cursor.description_ext()
```

Returns

Returns a description of the columns in the result set.

4.6.6.4 execute(operation[, parameters]) Method

Prepares and executes a database operation.

Syntax

```
cursor.execute(<operation>[, <parameters>])
```

Parameters

Recommended Type	Name	Description
String	`<operation>`	A query or command.
Tuple	`<parameters>`	The parameters for the specified procedure. The ability to specify parameters by using keywords is not defined in PEP 249.

Returns

Return values are not defined.

Example

Execute a single statement that does not return a result set:

```
cursor = conn.cursor()
cursor.execute("CREATE TABLE T1 (ID INTEGER PRIMARY KEY, C2 VARCHAR(255))")
cursor.close()
```

Use question mark parameter binding to insert values into the T1 table created above. The parameter values are supplied as a Python sequence and can be literal values or variable names. This example uses literal values:

```
sql = 'INSERT INTO T1 (ID, C2) VALUES (?, ?)'
cursor = conn.cursor()
cursor.execute(sql, (1, 'hello'))
# returns True
cursor.execute(sql, (2, 'hello again'))
# returns True
cursor.close()
```

Use named parameter binding to insert values into the T1 table. The values are supplied as a Python dictionary, and this example uses variable names.

```
sql = 'INSERT INTO T1 (ID, C2) VALUES (:id, :c2)'
cursor = conn.cursor()
id = 3
c2 = "goodbye"
cursor.execute(sql, {"id": id, "c2": c2})
# returns True
cursor.close()
```

Loop over the rows of the result set.

```
sql = 'SELECT * FROM T1'
cursor = conn.cursor()
cursor.execute(sql)
# returns True
for row in cursor:
    print(row)
```

4.6.6.5 executemany(operation, list_of_parameters[, batcherrors=False])

Prepares and executes a database operation for multiple sets of parameters.

Syntax

```
cursor.executemany(<operation>, <list_of_parameters>[, <batcherrors> = {True |
False} ])
```

Parameters

Recommended Type	Name	Description
String	`<operation>`	Specifies a query or command.
Tuple of Tuple	`<list_of_parameters>`	Specifies the parameters for the defined procedure.
Boolean	`<batcherrors>`	Toggles optional batch error reporting.

Returns

When `batcherrors` is set to False, executemany returns a tuple of rows affected counts corresponding to the results of execution with each set of parameters. When `batcherrors` is set to True, the returned value is not defined and the rows affected counts can be retrieved by calling the cursor.getrowsaffectedcounts() method.

batcherrors Parameters

The method cursor.executemany() has an optional Boolean parameter `batcherrors` that changes its behavior to communicate how many rows were affected by each row of input parameters, all of the errors that occurred in a multi-error execution case, and which rows caused the errors.

If errors occur during the call, a `dbapi.ExecuteManyError` exception is raised with the following characteristics:

- It is a dbapi.Error.
- Its errorcode is set to 0.
- Its error text is set to the following:

  ```
  An error occured during the executemany() execution. See errors for details
  ```
- It has the `errors` field, which is a tuple of dbapi.ExecuteManyErrorEntry. Each entry represents an error that occurred during the call.

Each `dbapi.ExecuteManyErrorEntry` entry:

- Is a dbapi.Error.
- Has its error code and error text set to the specified error from the statement or directly in the Python layer.
- Has the member rownumber (integer), which represents the row that caused the error.
  This is 1-based, or zero if no row caused the error. For example, an application programming error.

Example

Create a table named T1, then use executemany to insert multiple rows. The parameter values are supplied as an array of Python sequences:

```
cursor = conn.cursor()
cursor.execute("CREATE TABLE T2 (ID INTEGER PRIMARY KEY, C2 VARCHAR(255))")
sql = 'INSERT INTO T2 (ID, C2) VALUES (?, ?)'
cursor.executemany(sql, ((1, 'hello'), (2, 'hello again')))
cursor.close()
```

Loop over the rows in the result set:

```
sql = 'SELECT * FROM T2'
cursor = conn.cursor()
cursor.execute(sql)
# returns True
for row in cursor:
    print(row)
```

4.6.6.6 executemanyprepared(list_of_parameters) Method

Executes a previously prepared database operation in a batch format.

Syntax

```
cursor.executemanyprepared(<list_of_parameters>)
```

Parameters

Recommended Type	Name	Description
Tuple of Tuple	`<list_of_parameters>`	Specifies the parameters for the previously prepared operation.

Returns

Returns a tuple of the rows affected counts corresponding to the results of the execution with each set of parameters.

Remarks

Raises an exception if cursor.prepare() has not been called or if another query has been executed on the cursor since the last prepare.

Example

The following example creates a new cursor with a prepared operation:

```
sql = 'INSERT INTO T2 (ID, C2) VALUES (?, ?)'
cursor = conn.cursor()
cursor2 = cursor.prepare(sql, newcursor=True)
cursor2.executemanyprepared([(1, 'hello'), (2, 'hello again'), (3, 'goodbye')])
```

Related Information

prepare(operation[, newcursor=False]) Method [page 718]

4.6.6.7 executeprepared([parameters]) Method

Executes a previously prepared database operation.

Syntax

```
cursor.executeprepared([<parameters>])
```

Parameters

Recommended Type	Name	Description
Tuple of Tuple	`<parameters>`	Specifies the parameters for the previously prepared operation.

Returns

Returned values are not defined.

Remarks

Raises an exception if cursor.prepare() has not been called or if another query has been executed on the cursor since the last prepare.

Unlike the cursor.execute() method, parameters cannot be specified with keywords.

Example

The following example prepares an operation on an existing cursor:

```
sql = 'CREATE TABLE T2 (ID INTEGER PRIMARY KEY, C2 VARCHAR(255))'
cursor = conn.cursor()
cursor.prepare(sql)
cursor.executeprepared()
```

Related Information

prepare(operation[, newcursor=False]) Method [page 718]

4.6.6.8 fetchall() Method

Fetches all of the remaining rows of a query result as an array of sequences.

Syntax

```
cursor.fetchall()
```

Returns

Returns an array of all remaining rows in the query result.

Remarks

The `uselob` parameter cannot be used with `fetchall`, use the `fetchone` method to fetch LOB data as LOB class objects.

An error is returned if the previous call to an execute method did not produce a result set or if a call has not been issued yet.

Example

Use the following example to create a table:

```
cursor = conn.cursor()
cursor.execute("CREATE TABLE T2 (ID INTEGER PRIMARY KEY, C2 VARCHAR(255))")
sql = 'INSERT INTO T2 (ID, C2) VALUES (?, ?)'
cursor.executemany(sql, ((1, 'hello'), (2, 'hello again'), (3, 'goodbye' ))
cursor.close()
```

Use the following example to fetch all of the rows at once from the created table:

```
sql = "SELECT * FROM T2"
cursor = conn.cursor()
cursor.execute(sql)
rows = cursor.fetchall()
print(rows)
# prints [(1, 'hello'), (2, 'hello again'), (3, 'goodbye')]
```

4.6.6.9 fetchmany([size=cursor.arraysize]) Method

Fetches the next set of rows from a query result as an array of sequences.

Syntax

```
cursor.fetchmany([size=cursor.arraysize])
```

Returns

Returns the next set of result rows as an array. The size of the returned array is min(size, n) where n is the total number of rows remaining in the result set.

Remarks

The returned array may contain fewer than the `<size>` of the rows if there are less than the `<size>` of the rows remaining in the result set.

The `uselob` parameter cannot be used with fetchmany, instead use the fetchone method to fetch LOB data as LOB class objects.

An error is returned if the previous call to an execute method did not produce a result set or if a call has not been issued yet.

Example

Use the following example to create a table:

```
cursor = conn.cursor()
cursor.execute("CREATE TABLE T2 (ID INTEGER PRIMARY KEY, C2 VARCHAR(255))")
sql = 'INSERT INTO T2 (ID, C2) VALUES (?, ?)'
cursor.executemany(sql, ((1, 'hello'), (2, 'hello again'), (3, 'goodbye'))
cursor.close()
```

Use the following example to fetch multiple rows at once from the created table:

```
sql = "SELECT * FROM T2"
cursor = conn.cursor()
cursor.execute(sql)
rows = cursor.fetchmany(2)
print(rows)
# prints [(1, 'hell'), (2, 'hello again')]
rows = cursor.fetchmany(2)
print(rows)
# prints [(3, 'goodbye')]
```

4.6.6.10 fetchone([uselob]) Method

Fetches the next row of a result set.

Syntax

```
cursor.fetchone([uselob])
```

Parameters

Recommended Type	Name	Description

Boolean	`uselob`	Controls how LOB data is returned. If `uselob` is set to TRUE, then objects of the LOB class are returned in the result set for CLOBs/NCLOBs/BLOBs instead of the actual values. If you do not specify `uselob` as True, it leads to a cut-off of the actual value which results in application errors.
		
		The following example displays an out-of-date application code that must be updated to match an underlying CLOB column that contains very large JSON-formatted values:
		
		`params = controlTenantCur.fetchone()`
		`paramDict = json.loads(params[0])`
		
		The following example displays the updated application code:
		
		`params = controlTenantCur.fetchone(uselob=True)`
		`paramDict = json.loads(params[0].read())`
		
		The `uselob` parameter is not defined in PEP 249.

Returns

Returns the next row, or None when there is no further data available.

Remarks

An error is returned if the previous call to the execute method did not produce a result set or if a call has not been issued yet.

Example

Fetch a single row.

```
id = 2
sql = 'SELECT * FROM T1 WHERE ID = :id'
cursor = conn.cursor()
cursor.execute(sql, {"id": id})
row = cursor.fetchone()
print(row)
(2, 'hello again')
```

4.6.6.11 getpacketsize() Method

Returns the network packet size used by the cursor.

Syntax

```
cursor.getpacketsize()
```

Returns

Returns the network packet size used by the cursor, in bytes.

Related Information

Packet Size [page 821]

4.6.6.12 get_resultset_holdability() Method

Retrieves the result set holdability.

Syntax

```
cursor.get_resultset_holdability()
```

Returns

Returns a value that indicates whether the cursor for a statement is held or rolled back after the transaction is committed:

0	CURSOR_HOLD_OVER_COMMIT
1	CURSOR_CLOSE_ON_COMMIT_OR_ROLLBACK
2	CURSOR_HOLD_OVER_ROLLBACK
2	CURSOR_HOLD_OVER_ROLLBACK This value is only supported for SAP HANA.
3	CURSOR_HOLD_OVER_ROLLBACK_AND_COMMIT This value is only supported for SAP HANA.

Remarks

A held cursor is still valid after a commit or rollback, and the application is able to fetch more rows or result sets from it. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources.

Support for holding a cursor over rollback or over commit and rollback is available in SAP HANA servers beginning with SAP HANA 2.0 SPS 04.

An exception is raised if there is no connection.

Related Information

set_resultset_holdability(holdability) Method [page 727]

4.6.6.13 getrowsaffectedcounts() Method

Returns the count of affected rows for the previous call to executemany(..., batcherrors=True). This method fails if executemany with batcherrors=True was not previously called on the cursor.

Syntax

```
cursor.getrowsaffectedcounts()
```

Returns

Tuple of rows affected counts for the previous call to executemany(..., batcherrors=True).

> **Note**
>
> If an error occurs while processing a row, the rows affected count value for that row is zero (0).
>
> This method raises an exception if there is no previous method call on the cursor or if the previous method is not one of the following calls to executemany():
>
> - executemany(<operation>, <sequence_of_parameter_rows>, batcherrors=True)
> - executemany(<sequence_of_operations>, batcherrors=True)

Example

The following example is based on the below table:

```
cursor = conn.cursor()
cursor.execute("create table testtable(a tinyint unique)")
cursor.close()
```

Case 1: Prepared execute of a statement with a batch of input parameter rows (BATCH/BULK INSERT/ UPDATE/DELETE/SELECT):

```
cursor = conn.cursor()
try:
    cursor.executemany("insert into testtable(a) values(?)",[(1,), (2,)],
batcherrors=True)
    rowsaffected = cursor.getrowsaffectedcounts()
    # check rowsaffected (should be = (1, 1))
    # Next line should result in a dbapi.ExecuteManyError being raised
    cursor.executemany("insert into testtable(a) values(?)", [(3,), (4,), (3,),
(5,), (3,)], batcherrors=True)
except dbapi.ExecuteManyError as e:
    rowsaffected = cursor.getrowsaffectedcounts()
    # check rowsaffected (should be (1, 1, 0, 1, 0))
    for error_entry in e.errors:
        # process error by checking error_entry.errorcode, error_entry.errortext
and error_entry.rownumber
cursor.close()
```

Case 2: Execute a list of direct statements:

```
cursor = conn.cursor()
try:
    cursor.executemany(["insert into testtable(a) values(20)",
                        "insert into testtable(a) values(30)"],
                        batcherrors=True)
    rowsaffected = cursor.getrowsaffectedcounts()
    # check rowsaffected (should be = (1, 1))
    # next line should result in a dbapi.ExecuteManyError being raised
    cursor.executemany(["insert into testtable(a) values(40)",
                        None], # Not a string
                        batcherrors=True)
except dbapi.ExecuteManyError as e:
    rowsaffected = cursor.getrowsaffectedcounts()
```

```
    # Check rowsaffected
    for error_entry in e.errors:
        # process error by checking error_entry.errorcode, error_entry.errortext
and error_entry.rownumber
cursor.close()
```

Case 3: Use getrowsaffectedcounts() outside of executemany(batcherrors=True):

```
cursor = conn.cursor()
try:
    # Next line should result in a dbapi.ProgrammingError
    rowsaffected = cursor.getrowsaffectedcounts()
except dbapi.Error as e:
    # process error
cursor.close()
cursor = conn.cursor()
try:
    cursor.execute("insert into testtable(a) values(60)")
    # Next line should result in a dbapi.ProgrammingError
    rowsaffected = cursor.getrowsaffectedcounts()
except dbapi.Error as e:
    # process error
cursor.close()
cursor = conn.cursor()
try:
    cursor.executemany("insert into testtable(a) values(?)",
                       [(150,), (200,)])
    # Next line should result in a dbapi.ProgrammingError
    rowsaffected = cursor.getrowsaffectedcounts()
except dbapi.Error as e:
    # process error
cursor.close()
```

4.6.6.14 getwarning() Method

Retrieves a warning object if there is a warning from the previous execution.

Syntax

```
cursor.getwarning()
```

Returns

Returns a warning object.

Related Information

haswarning() Method [page 716]

4.6.6.15 haswarning() Method

Checks whether there are any warnings from the previous execution.

Syntax

```
cursor.haswarning()
```

Returns

Returns true if at least one warning is found.

Related Information

getwarning() Method [page 715]

4.6.6.16 nextset() Method

Makes the cursor skip to the next result set and closes the current result set.

Syntax

```
cursor.nextset()
```

Returns

Returns True if successful or None if there is no further result set to skip to.

Remarks

Moving between result sets is forward-only. After calling nextset(), the current result set is closed and can no longer be accessed.

Example

Use the following example to move the cursor to the next set of results:

```
cursor = conn.cursor()
cursor.execute("CREATE PROCEDURE P1 ()\n" +
                           "AS BEGIN\n" +
                           "SELECT * FROM T1;\n" +
                           "SELECT 1, 2, 3 FROM DUMMY;\n" +
                          "END;")
cursor.execute("call P1")
rows1 = cursor.fetchall()
print(rows1)
# prints [(1, 'hello'), (2, 'hello again')]
cursor.nextset()
# returns True
rows2 = cursor.fetchall()
print(rows2)
# prints [(1, 2, 3)]
cursor.nextset()
# returns None (no more result sets)
```

4.6.6.17 parameter_description() Method

Retrieves a sequence of parameter descriptions.

Syntax

```
cursor.parameter_description()
```

Returns

Returns parameter descriptions.

4.6.6.18 prepare(operation[, newcursor=False]) Method

Prepares a database operation.

Syntax

```
prepare(<operation>[, newcursor= { True | False } ])
```

Parameters

Recommended Type	Name	Description
String	`<operation>`	Specifies a query or command.
Boolean	`newcursor`	Controls whether the operation is prepared on the current cursor or if a new cursor is returned on which the statement has been prepared. The default value for `newcursor` is `False`.

Returns

If the value for `newcursor` is `True`, the statement returns a new cursor on which the operation has been prepared without modifying the original cursor. Otherwise, the returned value is undefined.

Example

The following example prepares an operation on an existing cursor:

```
sql = 'CREATE TABLE T2 (ID INTEGER PRIMARY KEY, C2 VARCHAR(255))'
cursor = conn.cursor()
cursor.prepare(sql)
cursor.executeprepared()
```

The following example creates a new cursor with a prepared operation:

```
sql = 'INSERT INTO T2 (ID, C2) VALUES (?, ?)'
cursor = conn.cursor()
cursor2 = cursor.prepare(sql, newcursor=True)
cursor2.executemanyprepared([(1, 'hello'), (2, 'hello again'), (3, 'goodbye')])
```

4.6.6.19 scroll(value[, mode]) Method

Scrolls through a result set.

Syntax

```
cursor.scroll(<value>[, mode])
```

Parameters

Recommended Type	Name	Description
Integer	`<value >`	Specify the number of rows to skip.
String	`mode`	Specify `absolute` or `relative`.

Returns

Throws an exception if the result set is not scrollable or if the cursor position is out of range.

Example

Skip one row in the result set:

```
scroll(1)
```

Skips two rows in the result set:

```
scroll(2)
```

Go to the first row in the result set:

```
scroll(1, 'absolute')
```

4.6.6.20 server_cpu_time() Method

Returns the server CPU processing time.

Syntax

```
Cursor.server_cpu_time()
```

Returns

The server CPU processing time, in microseconds. This value does not include network communication time.

Remarks

If the cursor is executed but not fetched, then the returned value is only for the most recent execution. If the cursor is fetched, then the returned value is for the fetches since the beginning of the most recent result set. If nextset is called, then the returned value is for the new result set.

By default, the execution includes the fetch server CPU processing time the first 32 rows. If fetching more rows requires additional fetch requests to the server, then the fetch server CPU processing time is a non-zero value and calculated from the beginning of the current result set.

If connected to an SAP HANA 2.0 SPS 03 server or earlier, `server_cpu_time()` always returns 0.

Example

Create the following table:

```
cursor = conn.cursor()
cursor.execute("CREATE TABLE TEST_DIAG (A INTEGER PRIMARY KEY)")
```

Example 1

Run the following INSERT statement:

```
cursor.execute("INSERT INTO TEST_DIAG VALUES(?)", (123))
```

Run the following method to retrieve the server CPU processing time from the INSERT statement:

```
cpuTime = cursor.server_cpu_time()
```

Example 2

Run the following SELECT statement:

```
cursor.execute("SELECT * FROM TEST_DIAG")
cursor.fetchall()
```

Run the following method to retrieve the fetch server CPU processing time (not including the execute server CPU processing time) from the SELECT statement:

```
cpuTime = cursor.server_cpu_time()
```

4.6.6.21 server_memory_usage() Method

Returns the server memory usage.

Syntax

```
Cursor.server_memory_usage()
```

Returns

The server memory usage, in bytes.

Remarks

If the cursor is executed but not fetched, then the returned value is only for the most recent execution. If the cursor is fetched, then the returned value is for the fetches since the beginning of the most recent result set. If nextset is called, then the returned value is for the new result set.

By default, the execution includes the memory usage for the first 32 rows. If fetching more rows requires additional fetch requests to the server, then the fetch memory usage is a non-zero value and calculated from the beginning of the current result set.

`server_memory_usage()` returns 0 if you are connected to an SAP HANA 2.0 SPS 03 server or earlier or unless memory_tracking and resource_tracking is enabled in the resource_tracking section of `global.ini` by setting enable_tracking and memory_tracking to "on". For performance reasons, resource tracking is disabled by default.

Example

Ensure that monitoring is enabled in the resource_tracking section of `global.ini` by setting enable_tracking and memory_tracking to "on". For performance reasons, resource tracking is disabled by default.

Create the following table:

```
cursor = conn.cursor()
cursor.execute("CREATE TABLE TEST_DIAG (A INTEGER PRIMARY KEY)")
```

Example 1

Run the following INSERT statement:

```
cursor.execute("INSERT INTO TEST_DIAG VALUES(?)", (123))
```

Run the following method to retrieve the server memory usage from the INSERT statement:

```
memoryUsage = cursor.server_memory_usage()
```

Example 2

Run the following SELECT statement:

```
cursor.execute("SELECT * FROM TEST_DIAG")
cursor.fetchall()
```

Run the following method to retrieve the fetch server memory usage (not including the execute server memory usage) from the SELECT statement:

```
memoryUsage = cursor.server_memory_usage()
```

4.6.6.22 server_processing_time() Method

Returns the elapsed server processing time.

Syntax

```
Cursor.server_processing_time()
```

Returns

The elapsed server processing time, in microseconds. This value does not include network communication time.

Remarks

If the cursor is executed but not fetched, then the returned value is only for the most recent execution. If the cursor is fetched, then the returned value is for the fetches since the beginning of the most recent result set. If nextset is called, then the returned value is for the new result set.

By default, the execution includes the fetch server processing time for the first 32 rows. If fetching more rows requires additional fetch requests to the server, then the server processing time is a non-zero value and calculated from the beginning of the current result set.

Example

Create the following table:

```
cursor = conn.cursor()
cursor.execute("CREATE TABLE TEST_DIAG (A INTEGER PRIMARY KEY)")
```

Example 1

Run the following INSERT statement:

```
cursor.execute("INSERT INTO TEST_DIAG VALUES(?)", (123))
```

Run the following method to retrieve the server processing time from the INSERT statement:

```
processingTime = cursor.server_processing_time()
```

Example 2

Run the following SELECT statement:

```
cursor.execute("SELECT * FROM TEST_DIAG")
cursor.fetchall()
```

Run the following method to retrieve the fetch server processing time (not including the execute server processing time) from the SELECT statement:

```
processingTime = cursor.server_processing_time()
```

4.6.6.23 setcommandinfo(command-info, line-number) Method

Sets an arbitrary string and integer as command information to be sent to the database along with the next command, `execute`.

Syntax

```
cursor.setcommandinfo(<command-info>, <line-number>)
```

Parameters

Recommended Type	Name	Description
String	`<command-info>`	Specifies the command information text to be sent along with the next command. This is typically a source file name.
Integer	`<line-number>`	Specifies the command information number to be sent along with the next command. This is typically a line number.

Returns

Always returns None.

Remarks

Command information set with this method is sent along with the next `execute` (or `executemany`, `executeprepared`, etc.), not at the time `setcommandinfo` is called.

Specify None as the first argument to reset the Cursor's command information. If `setcommandinfo` has been called since the previous `execute`, the values provided are cleared from the Cursor and no command information is sent with the next `execute`.

4.6.6.24 setfetchsize(value) Method

Sets the size of prefetched result sets.

Syntax

```
cursor.setfetchsize(<value>)
```

Parameters

Recommended Type	Name	Description
Integer	`<value>`	Specifies the number of rows to fetch from the result set.

Returns

Always returns none.

Remarks

Specify 0 to reset this method to the default. If there is no result set open, then the method still returns none.

4.6.6.25 setpacketsize(value) Method

Sets the network packet size used by the cursor.

Syntax

```
cursor.setpacketsize(<value>)
```

Parameters

Recommended Type	Name	Description
Integer	<value>	Sets the network packet size used by the cursor.

Returns

Always returns None.

Remarks

The default and minimum packet size used by the client is 1MB (1048576 bytes). Passing a value lower than this to setpacketsize(value) resets the packet size back to 1MB.

Example

Use the following example to set the packet size to 10 bytes, resetting the packet size to 1MB:

```
cursor = conn.cursor()
cursor.setpacketsize(2000000)
print(cursor.getpacketsize())
# prints 2000000
cursor.setpacketsize(10)
print(cursor.getpacketsize())
# prints 1048576, packet size has been reset to default
```

Related Information

Packet Size [page 821]

4.6.6.26 set_resultset_holdability(holdability) Method

Controls the behavior of the cursor on commit.

Syntax

```
cursor.set_resultset_holdability(<holdability>)
```

Parameters

Recommended Type	Name	Description
Integer	`<holdability>`	Specifies whether the cursor for a statement is held or rolled back after the transaction is committed. A held cursor is still valid after a commit or rollback, and the application is able to fetch more rows or result sets from it. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources. Support for holding a cursor over rollback or over commit and rollback is available in SAP HANA servers beginning with SAP HANA 2.0 SPS 04. 0 CURSOR_HOLD_OVER_COMMIT 1 CURSOR_CLOSE_ON_COMMIT_OR_ROLLBACK 2 CURSOR_HOLD_OVER_ROLLBACK This value is only supported for SAP HANA. 3 CURSOR_HOLD_OVER_ROLLBACK_AND_COMMIT This value is only supported for SAP HANA.

Remarks

An exception is raised if there is no connection.

Related Information

get_resultset_holdability() Method [page 712]

4.6.7 LOB Class

Streams data to or from SAP HANA.

Remarks

Documentation is provided for methods that are not part of PEP 249 or whose behavior differs from PEP 249.

Methods

Method	Description	Differences from PEP 249
close() Method [page 729]	Closes the LOB object.	Not defined in PEP 249.
find(object, length[, position]) Method [page 730]	Returns the offset of the specified pattern in the LOB object.	Not defined in PEP 249.
read([size][, position]) Method [page 730]	Returns the specified amount of data from the LOB object.	Not defined in PEP 249.
write(object) Method [page 731]	Writes the data to the LOB object.	Not defined in PEP 249.

Example

Add an image to an SAP HANA database:

```
# get an image from the file system
with open('image.png', 'rb') as f:
     img = f.read()
# write to the database
in_blob = dbapi.LOB()
id = 2
conn.setautocommit(False)
```

```
cursor.execute("update T1 set PHOTO = :img WHERE id =:id", {"img": in_blob,
"id": id})
in_blob.write(data=img)
in_blob.close()
conn.commit()
cursor.close()
```

Read the image back out of the database. Use of the `read()` method to fetch a portion of the blob at a time.

```
# read it into a stream
conn.setautocommit(True)
cursor = conn.cursor()
cursor.execute("select PHOTO from T1 where id = :id", {"id": id})
out_blob = cursor.fetchone(True)[0]
out_img = bytearray()
while True:
    data = out_blob.read(size=200)
    if data is None:
        print("Done.")
        break
    else:
        print("Reading more data...")
        out_img.extend(data.tobytes())
```

4.6.7.1 close() Method

Closes the LOB object.

Syntax

```
lob.close()
```

Returns

True or false to indicate whether the LOB object was closed successfully.

4.6.7.2 find(object, length[, position]) Method

Returns the offset of the specified pattern in the LOB object.

Syntax

```
lob.find(<object>, <length>[, <position>])
```

Parameters

Recommended Type	Name	Description
String	`<object>`	Specifies the pattern to search for.
Integer	`<length>`	Specifies how much of the pattern to use (for example, search using 10 characters of a 20 character string). The maximum length is 256 characters.
Integer	`<position>`	Specifies the position to start searching the data from. The default value is the current position.

Returns

The offset of the result in the LOB.

4.6.7.3 read([size][, position]) Method

Returns the specified amount of data from the LOB object.

Syntax

```
lob.read([<size>][, <position>)
```

Parameters

Recommended Type	Name	Description
Integer	`<size>`	Specifies the amount of data to read. The default value is the length of the LOB.
Integer	`<position>`	Specifies the position to start reading data from. The default value is the current position.

Returns

The requested data.

4.6.7.4 write(object) Method

Writes the specified data to the LOB object.

Syntax

```
lob.write(<object>)
```

Parameters

Recommended Type	Name	Description
String for CLOB or NCLOB Binary for BLOB	`<object>`	Specifies the object to be written to the LOB.

Returns

The length of the value written to the LOB.

4.6.8 SQLAlchemy Dialect for SAP HANA

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

The SQLAlchemy Dialect for SAP HANA is an open-source project hosted at GitHub that is actively maintained by SAP SE, and is not part of a licensed SAP HANA edition or option. It is provided under the terms of the project license.

Only the hdbcli dialect can connect to SAP HANA Cloud.

Related Information

https://github.com/SAP/sqlalchemy-hana/blob/master/LICENSE
https://github.com/SAP/sqlalchemy-hana/blob/master/README.rst
https://github.com/SAP/sqlalchemy-hana

4.7 Ruby Application Programming

SAP HANA supports two separate packages in the SAP HANA for Ruby project.

Native Ruby Driver

> ⚠ Caution
>
> The use of `gem` to download the hanaclient and activerecord-hanaclient-adapter gems from the public RubyGems repository is not currently supported and represents a significant security risk. Always execute the `gem` command from the local HANA Client install location and specify the gem filename.

hanaclient This package is a low-level driver that allows Ruby code to interface with SAP HANA databases. This package provides a Ruby wrapping over the interface exposed by the SAP HANA C++ API. This package is written in C++ and is available as pre-compiled gems for Microsoft Windows, Linux, and macOS. This package is a prerequisite for all other Ruby interfaces.

ActiveRecord Adapter

activerecord-hanaclient-adapter	This package is an adapter that allows ActiveRecord to communicate with the database server. ActiveRecord is an object-relational mapper that is part of the Ruby on Rails web development framework. This package is written in pure Ruby, and it is available in gem format. This adapter uses and is dependent on the hanaclient gem.

4.7.1 Configure Ruby for SAP HANA

Use `gem` to install SAP HANA for Ruby.

Context

> ⚠ Caution
>
> The use of `gem` to download the hanaclient and activerecord-hanaclient-adapter gems from the public RubyGems repository is not currently supported and represents a significant security risk. Always execute the `gem` command from the local HANA Client install location and specify the gem filename.

Procedure

1. Install the native Ruby driver (the hanaclient Ruby gem) from the main `/hdbclient` installation directory:

   ```
   cd ruby/hanaclient
   ```

   ```
   gem install hanaclient*.gem
   ```

2. Optionally, install the ActiveRecordAdapter from the main `/hdbclient` installation directory:

   ```
   cd ruby/activerecord-hana-client-adapter
   ```

   ```
   gem install activerecord-hanaclient-adapter*.gem
   ```

4.7.2 Configure Ruby on Rails Support

Rails is a web development framework written in the Ruby language and that is useful for web application development. Add SAP HANA to the set of database management systems supported by Rails.

Procedure

1. Install the Ruby interpreter on your system.
2. Install RubyGems to simplify the installation of Ruby packages. The Ruby on Rails download page recommends which version to install.
3. Install Rails and its dependencies by running the following command:

   ```
   gem install rails
   ```

4. Install the Ruby Development Kit (DevKit).
5. Install the ActiveRecord support (activerecord-hanaclient-adapter) from the main `/hdbclient` directory:

   ```
   cd ruby/activerecord-hanaclient-adapter
   ```

   ```
   gem install activerecord-hanaclient-adapter
   ```

6. Add SAP HANA to the set of database management systems supported by Rails.
   a. Configure a database by creating a `hanaclient.yml` file in the Rails `configs\databases` directory. If you have installed Ruby in the `\Ruby` directory and you have installed version 3.1.13 of Rails, then the path to this file is `\Ruby\lib\ruby\gems\1.9.1\gems\railties-3.1.13\lib\rails\generators\rails\app\templates\config\databases`. The contents of this file are as follows:

   ```
   #
   # SAP HANA database configuration
   #
   # The specified username and
   # password should permit sufficient access to the database
   # to create tables.
   #
   development:
     adapter: hanaclient
     server: <hostname_of_HANA_server>
     database: <HANA_database_name>_development
     port: <port_of_HANA_server>
     username: <username>
     password: <password>
   ```

   The `hanaclient.yml` file provides a template for creating `database.yml` files in Rails projects. The following database options can be specified:

adapter	(Required, no default) This option must be set to hanaclient to use the ActiveRecord adapter.
database	(Optional) This option corresponds to `databaseName` in a connection string.

server	(Required, no default) Specifies the hostname or IP address of the SAP HANA server.
username	This option corresponds to the userid (UID) in a connection string.
password	This option corresponds to the password (PWD) in a connection string.
connection_properties	(Optional) This option defines additional connection properties and is an arbitrary string that is appended to the end of the connection string.
port	(Required) Specifies the connection port.

b. Update the Rails `app_base.rb` file. Using the same assumptions in the previous step, this file is located in the path `\Ruby\lib\ruby\gems\2.4.0\gems\railties-5.1.4\lib\rails\generators\app_base.rb`. Edit the `app_base.rb` file and locate the following line:

```
DATABASES = %w( mysql oracle postgresql sqlite3 frontbase ibm_db
sqlserver )
```

Add hanaclient to the list as follows:

```
DATABASES = %w( hanaclient mysql oracle postgresql sqlite3 frontbase
ibm_db sqlserver )
```

Results

You have successfully configured Rails support.

4.7.2.1 Learning Rails

Use the tutorial on the Ruby on Rails website to learn about Rails development.

Prerequisites

Rails must be configured for use with SAP HANA.

Use the tutorial on the Ruby on Rails website and this task to become familiar with Rails development.

Procedure

1. In the tutorial, you are shown the command to initialize the **`blog`** project. Run the following command to initialize the **`blog`** project for use with SAP HANA:

   ```
   rails new blog -d hanaclient
   ```

2. After you create the **`blog`** application, switch to its folder to continue work directly in that application:

   ```
   cd blog
   ```

3. Edit the `Gemfile` file to include a *gem* directive for the SAP HANA ActiveRecord adapter. Add the new directive following the line below:

   ```
   gem 'hanaclient'
   gem 'activerecord-hanaclient-adapter'
   ```

4. Follow the remaining steps in the tutorial.

Results

You are now prepared for Ruby on Rails software development.

Related Information

http://guides.rubyonrails.org/getting_started.html#creating-the-blog-application

4.7.3 Run Ruby Examples

Configure the Ruby driver so that you can run the examples in the SAP HANA Ruby API reference.

Procedure

Run the following code before using the examples in the SAP HANA Ruby API reference.

```
@api = HANACLIENT::HANACLIENTInterface.new()
HANACLIENT::API.hanaclient_initialize_interface(@api)
@api.hanaclient_init()
@conn = @api.hanaclient_new_connection()
@api.hanaclient_connect(@conn,
"serverNode=localhost:30015;UID=SYSTEM;PWD=Password123;")

@api.hanaclient_execute_immediate(@conn, <<-end_sql)
CREATE TABLE "Products"
```

```
(
  "ID" integer NOT NULL,
  "Name" varchar(15) NOT NULL,
  "Description" varchar(30) NOT NULL,
  "Size" varchar(18) NOT NULL,
  "Color" varchar(18) NOT NULL,
  "Quantity" integer NOT NULL,
  "UnitPrice" decimal(15,2) NOT NULL,
  CONSTRAINT "ProductsKey" PRIMARY key ("ID")
)
end_sql

@api.hanaclient_execute_immediate(@conn, <<-end_sql)
INSERT INTO "Products" VALUES
(
  300,
  'Tee Shirt',
  'V-neck',
  'Medium',
  'Orange',
  54,
  14.00
);
end_sql

@api.hanaclient_commit(@conn)

#stmt = @api.hanaclient_prepare(@conn, "SELECT * FROM \"accounts\" WHERE \"id\"
= ?")
#res, param = @api.hanaclient_describe_bind_param(stmt, 0)
#param.set_value(1) #@api.hanaclient_bind_param(stmt, 0, param)
#@api.hanaclient_execute(stmt) #@api.hanaclient_fetch_next(stmt)
#print @api.hanaclient_get_column(stmt, 0) #@api.hanaclient_free_stmt(stmt)

@api.hanaclient_disconnect(@conn)
@api.hanaclient_free_connection(@conn)
@api.hanaclient_fini()
HANACLIENT::API.hanaclient_finalize_interface(@api)
```

4.7.4 Ruby Connection Properties

When connecting to an SAP HANA database using Ruby, you can specify DBCAPI connection properties or SQLDBC connection properties.

The following table lists the SAP HANA client-specific DBCAPI connection properties (property names are case insensitive):

Property	Value	Default	Description
`charset`	`<character-set>`	UTF-8	Specifies the character set. The possible values are ASCII, UTF-8, or UCS2.
`database`	`<name>`		Specifies the name of the database to connect to.

Property	Value	Default	Description
PWD	<password>		Specifies the password for the SAP HANA database user. This property is optional depending on the used authentication method.
serverNode	<host>:<port>[, <host>:<port>]		Specifies the string of host:port pairs, separated by a comma, of the SAP HANA servers to connect to.
UID	<user-ID>		Specifies the name of the SAP HANA database user. This property is optional depending on the used authentication method.

SQLDBC Connection Properties

The following table lists SQLDBC connection properties (property names are case insensitive) that can also be used with Ruby:

> **Note**
>
> A valid setting for a Boolean property can be: “1”, “TRUE”, “YES”, or “ON” for a TRUE setting or “0”, “FALSE”, “NO”, or “OFF” for a FALSE setting. The setting of a Boolean property is case-insensitive.

Property	Value	Default	Description
abapVarCharMode	Boolean	FALSE	Specifies that in ABAP VARCHAR mode, a single space and an empty string are considered the same.

Property	Value	Default	Description
`allowLocalCompress`	Boolean	TRUE	Determines whether or not connections from the same machine as the SAP HANA server are compressed. The `compress` parameter and/or the `indexserver.ini` > *session* > *compression* server configuration control whether or not network compression is enabled. **Note** If the connection uses a proxy, do not use `allowLocalCompress`=FALSE as the connection may not be compressed when required.
`allowReconnectOnSelect`	Boolean	OFF	Allows reconnects to happen for SELECT queries which fail during a receive. By default, this property is set to OFF, which allows for backward compatibility.

Property	Value	Default	Description
`bindAddress`	`<ip-address>`	NULL	Specifies that the provided address is used as the source address for the socket connection. By default, the address preferred by the OS is used. If setting `bindAddress` to a hostname, that hostname should only resolve to a single IP address. In the majority of network configurations, specifying `bindAddress` is not required. **Note** Only specify `bindAddress` if it is required to connect in your network configuration.
`chopBlanks`	Boolean	FALSE	Removes trailing blanks from character output.
`chopBlanksInput`	Boolean	FALSE	Removes trailing blanks from character input.
`communicationTimeout`	Timeout in milliseconds	0 (disabled)	Aborts communication attempts to the server after the specified timeout. If set to a non-zero value, the driver will wait for a maximum of the specified number of milliseconds for any request to the server to complete. If the time has expired and a response has not been received, the driver returns an error.

Property	Value	Default	Description
`compress`	Boolean	FALSE	Enables or disables network compression for the connection. The `indexserver.ini` > *session* > *compression* server configuration defaults to FALSE. Setting this parameter overrides the setting of the `indexserver.ini` > *session* > *compression* parameter. The `allowLocalCompress` property controls if clients running on the same machine as the server use network compression when compression is requested.
`connDownRollbackError`	Boolean	FALSE	Setting this property to TRUE specifies that if the database connection is down due to network or SAP HANA server problems, the client rolls back the last transaction and sends and error to the application. The application can resend the last transaction to the server after properly handling the error. The following conditions must be true: • The session is recoverable in the server • There is a written transaction (uncommitted operation) in the session • The client failed to send the last SQL statement, or the last statement was sent to the server successfully, but both auto-commit and DDL auto-commit in the client are off. By default, this property is set to FALSE.

Property	Value	Default	Description
`connectTimeout`	Timeout in milliseconds	0 (limited by the `nodeConnectTimeout` connection property)	Aborts connection attempts after the specified timeout. `connectTimeout` limits the overall time for the entire connection to try all server addresses specified. See `nodeConnectTimeout` for per-node timeout.
`currentSchema`	`<schema>`	CURRENTUSER	Sets the current schema, which is used for identifiers without a schema.
`cursorHoldabilityType`	NONE, COMMIT, ROLLBACK, COMMIT_AND_ROLLBACK	COMMIT	Specifies whether the cursor for a result set remains open after the transaction is committed or rolled back. If a cursor remains open, then the application is able to fetch more rows from it. NONE means the cursor is always closed after a commit or rollback, while COMMIT means the cursor remains valid after a commit, but not after rollback. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources. Support for holding a cursor over ROLLBACK or COMMIT_AND_ROLLBACK is available in SAP HANA servers beginning with SAP HANA 2.4.

Property	Value	Default	Description
`emptyTimestampIsNull`	Boolean	TRUE	When enabled, DATE, TIME, SECONDDATE, and TIMESTAMP values inserted as empty strings are returned as NULLs. When disabled, these values are returned as the following out-of-band values: • `DATE` - one day before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 00:00.000000000 UTC BC • `TIME` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `SECONDDATE` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `TIMESTAMP` - one hundred nanoseconds before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.999999900 UTC BC

Property	Value	Default	Description
`ignoreTopology`	Boolean	FALSE	Controls how the client uses the topology returned from the server. TRUE indicates to use the topology unless port-forwarding is detected and FALSE indicates to always ignore the topology. It is an error to specify both ignoreTopology=TRUE and siteType=PRIMARY or SECONDARY.
`isolationLevel`	TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE	TRANSACTION_READ_COMMITTED	Sets the isolation level for the connection.
`isoTimestampOutput`	`<string>`	FALSE	Displays whether or not 'T' is used as the separator between the Date and Time parts of a timestamp retrieved as a string: TRUE/FALSE. If FALSE, then a space character is used as the separator.
`key`	`<username>`	Empty	Specifies the key for the hdbuserstore.
`maxLazyDroppedStatements`	`<number>`	50	Controls the maximum number of queued statements to be dropped. Setting this value to 0 disables the lazy drop feature and causes a request to be sent to the server each time a statement is dropped by the application.
`maxVersion`	`<string>`		Specifies the maximum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.

Property	Value	Default	Description
`minVersion`	`<string>`		Specifies the minimum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.
`nodeConnectTimeout`	Timeout in milliseconds	60000	Aborts the connection attempt after the specified maximum timeout. The connection attempt typically fails immediately if the server cannot be found. If connection attempts to a particular server address are slow to fail, then using this connection property reduces the time before a successful connection by minimizing the time waiting to connect to individual addresses. On a host with multiple resolved IP addresses, the network timeout for each attempt to connect to an IP address will be equal to the `nodeConnectTimeout` divided by the number of resolved IP addresses. If this option is set to zero, the `connectTimeout` setting will be used. If both of the `nodeConnectTimeout` and `connectTimeout` are set to zero, then SQLDBC will use the system TCP/IP socket connect timeout to connect to the IP address.

Property	Value	Default	Description
`packetCaching`	Boolean	TRUE	Caching packets reduces the need to continually allocate and deallocate memory for packets. However, this increases the peak memory usage of the application, and for how long it stays at or near that peak.
`packetSize`	`<number>` in bytes	1048576 (1 MB)	Sets the maximum size of a request packet sent from the client to the server, in bytes. In some cases, request packets can be larger (for example, if a SQL statement or parameter value is larger than the packetSize). The minimum is 1 MB.
`packetSizeLimit`	`<number>` in bytes	2147483647 (2 GB)	Sets the maximum size for a request or reply packet, in bytes. If SQLDBC needs to allocate a packet larger than the limit, then it returns an error. If a reply packet is too large, then the connection is dropped. The minimum value is the packetSize value; the maximum value is 2 GB.

Property	Value	Default	Description
`prefetch`	Boolean	TRUE	Controls result set prefetching. When enabled, result set prefetch requests the next block of rows of a result set while the application is still processing the current block of rows. When fetching large result sets, result set prefetch can improve performance; however, it can also result in the client using more memory. In some cases, the server JobExecutor thread could be blocked from sending the reply for an indefinite period of time until the client receives the prefetch reply. If the result set contains LOB columns, then result set prefetch is always disabled.
`{ proxyHostname | proxy_host }`	`<host-name>`		Specifies the host name of the proxy server. The default proxy server is SOCKS5. If `proxyHttp` is set to true, the proxy server is assumed to be an HTTP proxy.
`proxyHttp`	Boolean	FALSE	When set to TRUE, enables HTTP proxy authentication and implies that ENCRYPT=TRUE. This property is only applicable if `proxyHostname` is specified and `webSocketURL` is not specified.

Property	Value	Default	Description
`{ proxyPassword | proxy_password }`	`<password>`		Specifies the password for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the Cloud Connector location ID value. The value must be entered as a base64 encoded value with a "1." prepended to the string.
`{ proxyPort | proxy_port }`	`<port-number>`		Specifies the port of the proxy server. The default is 1080.
`{ proxyScpAccount | proxy_scp_account }`	SAP BTP, Cloud Foundry Environment: `<location-ID>` SAP BTP, Neo Environment: `<subaccount>.<location-ID>`		Provides SAP Cloud Connector routing information to the Cloud Connector SOCKS proxy. In an SAP BTP, Cloud Foundry Environment, provide an optional `<locationID>` in plaintext. Internally, the value is converted to base64 encoding. This value overrides the `proxyPassword` connection property. In an SAP BTP, Neo Environment, either `proxyScpAccount` or `proxyUsername` must be specified. `<subaccount>` is mandatory and `<locationID>` is optional. Both values are entered as plaintext strings and are internally converted to base64 encoding. This value overrides the `proxyUsername` connection property.

Property	Value	Default	Description
`{ proxyUserName | proxy_userid }`	`<user-name>`		Specifies the user name for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the JSON Web Token (JWT) value. When using an SAP HANA Cloud Connector in the SAP BTP, Neo Environment, this field is used to provide the subaccount and location ID details. The values must be base64 encoded manually using the format `1.base64(subaccount).base64(locationID)`.
`reconnect`	Boolean	TRUE	Controls whether the system automatically reconnects to the database instance after a command timeout or when the connection is broken. Reconnecting restores the old state (for example, if no transaction was open).

Property	Value	Default	Description
`resolveHostName`	IPv4First IPv6First	IPv4First	Controls which addresses to use in a server address, especially if an address or hostname resolves to multiple IP addresses. In all cases, stop attempting further resolved addresses once a TCP/IP connection is successfully established. The following values can be used with `resolveHostName`: • IPv4First (default) - try all resolved addresses starting with IPv4 addresses. • IPv6First - try all resolved addresses starting with IPv6 addresses.
`sessionVariable:<key-name>`	`<session variable value>`	None except for APPLICATION and APPLICATIONUSER keys	Sets the session variable `<key-name>` to `<session variable value>`. For example, to set the session variable with key `APPLICATION`, use `sessionVariable:APPLICATION=MyApplication`.

Property	Value	Default	Description
`statementCacheSize`	`<number>`	0 (no statements cached)	Controls the maximum number of cached prepared statements. Caching SQL statements that are frequently prepared and dropped can improve performance. DDL can cause behavior changes when statement caching is enabled. **Note** If it is possible that the definition of objects referred to by the prepared statements may change because DDL is executed while you are connected, then it is recommended that you do not enable prepared statement caching.
`statementRoutingFailureBackoff`	Boolean	TRUE	Enables or disables the re-establishment of an internal connection to a specific host for a time period, after the internal connection fails during the preparation or execution of a statement routed to that host. The minimum time period is 1 minute. The time period is doubled after each subsequent internal connection establishment or usage failure for the same node, up to a maximum of 1 hour. Every successful internal connection re-establishment and usage resets the time period to the minimum value.

Property	Value	Default	Description
`statementRoutingWarnings`	Boolean	FALSE	Returns a SQL warning to the application with SQLCODE -11001 or -11003 that gives information about the routing failure of a prepared statement.
`tcpKeepAliveCount`	`<number-of-retries>` This value is ignored if it is 0.	5	Controls the number of KeepAlive packets sent to the server before the driver assumes that the connection is closed. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.
`tcpKeepAliveIdle`	`<seconds>` This value is ignored if it is 0.	200	Controls the number of seconds that the connection is idle before the driver sends a KeepAlive packet. If 0 is specified, this value is determined by the operating system.
`tcpKeepAliveInterval`	`<seconds>` This value is ignored if it is 0.	4	Controls the number of specified seconds before the driver sends an additional KeepAlive packet if there was no response to the previous KeepAlive packet. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.

Property	Value	Default	Description
`timestampPadding`	Boolean	TRUE	In a timestamp where the format contains a fraction, this property determines whether the fractional portion of the timestamp will be 7 or 9 digits. When disabled, the fractional portion is 7 digits to match how the server displays timestamps. Otherwise, it is 9 digits. **Note** ABAP timestamps are not affected regardless of the value of `timestampPadding`.
`traceFile`			Specifies the trace file name. Only the stdout and stderr values are supported.
`traceOptions`			Implicitly enables tracing for all SQLDBC-based client connections. See the HDB_SQLDBC_TRACEOPTS description in the **Environment Variables** topic for more information.
`webSocketURL` (SAP HANA Service only)	`<webSocketUrl>`		Specifies the WebSocket URL used to connect to a SAP HANA server. The connection property value cannot be empty. WebSockets connections have the following default connection property settings: • encrypt=TRUE • compress=TRUE • reconnect=FALSE

4.7.5 SAP HANA Ruby API Reference

The Ruby extension API permits the rapid development of SQL applications. This extension is built on a low-level interface called the SAP HANA C API.

To demonstrate the power of Ruby application development, consider the following sample Ruby program. It loads the Ruby extension, connects to the sample database, lists column values from the Products table, disconnects, and terminates.

```
begin
    require 'rubygems'
    gem 'hanaclient'
    unless defined? hanaclient
        require 'hanaclient'
    end
end
api = HANACLIENT::HANACLIENTInterface.new()
HANACLIENT::API.hanaclient_initialize_interface( api )
api.hanaclient_init()
conn = api.hanaclient_new_connection()
puts "Enter user ID"
myuid = STDIN.gets.chomp
puts "Enter password"
mypwd = STDIN.gets.chomp
api.hanaclient_connect( conn, "serverNode=<DSN-NAME>;UID="+myuid+";PWD="+mypwd )
stmt = api.hanaclient_execute_direct( conn, "SELECT * FROM Products" )
num_rows = api.hanaclient_num_rows( stmt )
num_rows.times {
    api.hanaclient_fetch_next( stmt )
    num_cols = api.hanaclient_num_cols( stmt )
    for col in 1..num_cols do
        info = api.hanaclient_get_column_info( stmt, col - 1 )
        unless info[3]==1 # Don't do binary
            rc, value = api.hanaclient_get_column( stmt, col - 1 )
            print "#{info[2]}=#{value}\n"
        end
    end
    print "\n"
}
api.hanaclient_free_stmt( stmt )
api.hanaclient_disconnect(conn)
api.hanaclient_free_connection(conn)
api.hanaclient_fini()
HANACLIENT::API.hanaclient_finalize_interface( api )
```

The first two rows of the result set output from this Ruby program are shown below:

```
ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28
UnitPrice=9.00
ID=301
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00
```

4.7.5.1 hanaclient_affected_rows Function

Returns the number of rows affected by the execution of the prepared statement.

Syntax

```
hanaclient_affected_rows ( <$stmt> )
```

Parameters

$stmt — A statement that was prepared and executed successfully in which no result set was returned. For example, an INSERT, UPDATE, or DELETE statement.

Returns

Returns a scalar value (the number of rows affected), or -1 on failure.

Example

```
affected = api.hanaclient_affected( stmt )
```

Related Information

hanaclient_execute Function [page 762]

4.7.5.2 hanaclient_bind_param Function

Binds a user-supplied buffer as a parameter to the prepared statement.

Syntax

```
hanaclient_bind_param ( <$stmt>, <$index>, <$param> )
```

Parameters

$stmt	A statement object returned by the successful execution of hanaclient_prepare.
$index	The index of the parameter. The number must be between 0 and hanaclient_num_params() - 1.
$param	A filled bind object retrieved from hanaclient_describe_bind_param.

Returns

Returns a scalar value that is 1 when successful or 0 when unsuccessful.

Example

```
stmt = api.hanaclient_prepare(conn, "UPDATE "Products"
     SET "ID" = "ID" + 1000
     WHERE "ID" >= ?" )
rc, param = api.hanaclient_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.hanaclient_bind_param( stmt, 0, param )
```

Related Information

hanaclient_describe_bind_param Function [page 759]

4.7.5.3 hanaclient_clear_error Function

Clears the last stored error code.

Syntax

```
hanaclient_clear_error ( <$conn> )
```

Parameters

$conn	A connection object returned from hanaclient_new_connection.

Returns

Returns nil.

Example

```
api.hanaclient_clear_error( conn )
```

Related Information

hanaclient_new_connection Function [page 775]

4.7.5.4 hanaclient_client_version Function

Returns the current client version.

Syntax

```
hanaclient_client_version ( )
```

Returns

Returns a scalar value that is the client version string.

Example

```
buffer = api.hanaclient_client_version()
```

4.7.5.5 hanaclient_commit Function

Commits the current transaction.

Syntax

```
hanaclient_commit ( <$conn> )
```

Parameters

$conn	The connection object that the commit operation is performed on.

Returns

Returns a scalar value that is 1 when successful or 0 when unsuccessful.

Example

```
rc = api.hanaclient_commit( conn )
```

Related Information

hanaclient_rollback Function [page 780]

4.7.5.6 hanaclient_connect Function

Creates a connection to a database server using the specified connection object and connection string.

Syntax

```
hanaclient_connect ( <$conn>, <$str> )
```

Parameters

$conn	The connection object created by hanaclient_new_connection.
$str	A valid connection string.

Returns

Returns a scalar value that is 1 if the connection is established successfully or 0 if the connection fails. Use hanaclient_error to retrieve the error code and message.

Example

```
require 'hanaclientwhere'
api = hanaclientwhere::hanaclientwhereInterface.new()
hanaclientwhere::API.hanaclient_initialize_interface( api )
api.hanaclient_init()
# Create a connection
conn = api.hanaclient_new_connection()
puts "Enter user ID"
myuid = STDIN.gets.chomp
puts "Enter password"
mypwd = STDIN.gets.chomp
# Establish a connection
status = api.hanaclient_connect( conn, "UID="+myuid+";PWD="+mypwd )
print "Connection status = #{status}\n"
```

Related Information

hanaclient_new_connection Function [page 775]
hanaclient_error Function [page 761]

4.7.5.7 hanaclient_describe_bind_param Function

Describes the bind parameters of a prepared statement.

Syntax

```
hanaclient_describe_bind_param ( <$stmt>, <$index> )
```

Parameters

$stmt	A statement prepared successfully using hanaclient_prepare.
$index	The index of the parameter. The number must be between 0 and hanaclient_num_params() - 1.

Returns

Returns a 2-element array where the first element is 1 if successful or 0 if unsuccessful, and the second element is a described parameter.

Remarks

This function describes prepared statement parameters. The type of prepared statement (stored procedure or DML), determines the amount of information provided. The direction of the parameters (input, output, or input-output) are always provided.

Example

```
stmt = api.hanaclient_prepare(conn, "UPDATE "Products"
     SET "Products".ID = "Products".ID + 1000
     WHERE "Products".ID >= ?" )
rc, param = api.hanaclient_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.hanaclient_bind_param( stmt, 0, param )
```

Related Information

hanaclient_bind_param Function [page 755]
hanaclient_prepare Function [page 778]

4.7.5.8 hanaclient_disconnect Function

Disconnects a database connection. All uncommitted transactions are rolled back.

Syntax

```
hanaclient_disconnect ( <$conn> )
```

Parameters

$conn	A connection object with a connection established by using hanaclient_connect.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
# Disconnect from the database
status = api.hanaclient_disconnect( conn )
print "Disconnect status = #{status}\n"
```

Related Information

hanaclient_connect Function [page 758]
hanaclient_new_connection Function [page 775]

4.7.5.9 hanaclient_error Function

Returns the last error code and message stored in the connection object.

Syntax

```
hanaclient_error ( <$conn> )
```

Parameters

$conn	A connection object returned from hanaclient_new_connection.

Returns

Returns a 2-element array where the first element is the SQL and the second element is an error message string.

For the error code, positive values are warnings, negative values are errors, and 0 is success.

Example

```
code, msg = api.hanaclient_error( conn )
print "Code=#{code} Message=#{msg}\n"
```

Related Information

hanaclient_error Function [page 761]

4.7.5.10 hanaclient_execute Function

Executes a prepared statement.

Syntax

```
hanaclient_execute ( <$stmt> )
```

Parameters

$stmt	A statement prepared successfully using hanaclient_prepare.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Remarks

Use hanaclient_num_cols to verify if the statement returned a result set.

Example

```
stmt = api.hanaclient_prepare(conn, "UPDATE "Products"
     SET "Products".ID = "Products".ID + 1000
     WHERE "Products".ID >= ?" )
rc, param = api.hanaclient_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.hanaclient_bind_param( stmt, 0, param )
rc = api.hanaclient_execute( stmt )
```

Related Information

hanaclient_prepare Function [page 778]

4.7.5.11 hanaclient_execute_direct Function

Executes the SQL statement specified by the string argument.

Syntax

```
hanaclient_execute_direct ( <$conn>, <$sql> )
```

Parameters

$conn	A connection object with a connection established by using hanaclient_connect.
$sql	A SQL string. The SQL string should not have parameters such as ?.

Returns

Returns a statement object or nil on failure.

Remarks

Use this function to prepare and execute a statement in one step. Do not use this function to execute a SQL statement with parameters.

Example

```
stmt = api.hanaclient_execute_direct( conn, "SELECT * FROM "Products"" )
rc = api.hanaclient_fetch_next( stmt )
rc, employeeID = api.hanaclient_get_column( stmt, 0 )
rc, managerID  = api.hanaclient_get_column( stmt, 1 )
rc, surname = api.hanaclient_get_column( stmt, 2 )
rc, givenName = api.hanaclient_get_column( stmt, 3 )
rc, departmentID = api.hanaclient_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "\n"
```

Related Information

hanaclient_fetch_absolute Function [page 765]
hanaclient_fetch_next Function [page 766]
hanaclient_num_cols Function [page 776]
hanaclient_get_column Function [page 771]

4.7.5.12 hanaclient_execute_immediate Function

Executes the specified SQL statement immediately without returning a result set. This function is useful for statements that do not return result sets.

Syntax

```
hanaclient_execute_immediate ( <$conn>, <$sql> )
```

Parameters

$conn	A connection object with a connection established by using hanaclient_connect.
$sql	A SQL string. The SQL string should not have parameters such as ?.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
rc = api.hanaclient_execute_immediate(conn, "UPDATE "Products"
     SET "Products".ID = "Products".ID + 1000
     WHERE "Products".ID >= 50" )
```

Related Information

hanaclient_fetch_absolute Function [page 765]

4.7.5.13 hanaclient_fetch_absolute Function

Moves the current row in the result set to the row number specified and then fetches the data from that row.

Syntax

```
hanaclient_fetch_absolute ( <$stmt>, <$row_num> )
```

Parameters

$stmt	A statement object that was executed by hanaclient_execute or hanaclient_execute_direct.
$row_num	The row number to be fetched. The first row is 1, the last row is -1.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
stmt = api.hanaclient_execute_direct( conn, "SELECT * FROM "Products"" )
# Fetch the second row
rc = api.hanaclient_fetch_absolute( stmt, 2 )
rc, employeeID = api.hanaclient_get_column( stmt, 0 )
rc, managerID  = api.hanaclient_get_column( stmt, 1 )
rc, surname = api.hanaclient_get_column( stmt, 2 )
rc, givenName = api.hanaclient_get_column( stmt, 3 )
rc, departmentID = api.hanaclient_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "\n"
```

Related Information

hanaclient_error Function [page 761]
hanaclient_execute Function [page 762]
hanaclient_execute_direct Function [page 763]
hanaclient_fetch_next Function [page 766]

4.7.5.14 hanaclient_fetch_next Function

Returns the next row from the result set. This function first advances the row pointer and then fetches the data at the new row.

Syntax

```
hanaclient_fetch_next ( <$stmt> )
```

Parameters

$stmt	A statement object that was executed by hanaclient_execute or hanaclient_execute_direct.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
stmt = api.hanaclient_execute_direct( conn, "SELECT * FROM "Products"" )
# Fetch the second row
rc = api.hanaclient_fetch_next( stmt )
rc, employeeID = api.hanaclient_get_column( stmt, 0 )
rc, managerID  = api.hanaclient_get_column( stmt, 1 )
rc, surname = api.hanaclient_get_column( stmt, 2 )
rc, givenName = api.hanaclient_get_column( stmt, 3 )
rc, departmentID = api.hanaclient_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "\n"
```

Related Information

hanaclient_error Function [page 761]
hanaclient_execute Function [page 762]
hanaclient_execute_direct Function [page 763]
hanaclient_fetch_absolute Function [page 765]

4.7.5.15 hanaclient_fini Function

Frees resources allocated by the API.

Syntax

hanaclient_fini ( )

Returns

Returns nil.

Example

```
# Disconnect from the database
api.hanaclient_disconnect( conn )
# Free the connection resources
api.hanaclient_free_connection( conn )
# Free resources the api object uses
api.hanaclient_fini()
# Close the interface
hanaclientwhere::API.hanaclient_finalize_interface( api )
```

Related Information

hanaclient_init Function [page 774]

4.7.5.16 hanaclient_free_connection Function

Frees the resources associated with a connection object.

Syntax

```
hanaclient_free_connection ( <$conn> )
```

Parameters

$conn	A connection object created by hanaclient_new_connection.

Returns

Returns nil.

Example

```
# Disconnect from the database
api.hanaclient_disconnect( conn )
# Free the connection resources
api.hanaclient_free_connection( conn )
# Free resources the api object uses
```

```
api.hanaclient_fini()
# Close the interface
hanaclientwhere::API.hanaclient_finalize_interface( api )
```

Related Information

hanaclient_new_connection Function [page 775]

4.7.5.17 hanaclient_free_stmt Function

Frees resources associated with a statement object.

Syntax

```
hanaclient_free_stmt ( <$stmt> )
```

Parameters

$stmt — A statement object returned by the successful execution of hanaclient_prepare or hanaclient_execute_direct.

Returns

Returns nil.

Example

```
stmt = api.hanaclient_prepare(conn, "UPDATE "Products"
     SET "Products".ID = "Products".ID + 1000
     WHERE "Products".ID >= ?" )
rc, param = api.hanaclient_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.hanaclient_bind_param( stmt, 0, param )
rc = api.hanaclient_execute( stmt )
rc = api.hanaclient_free_stmt( stmt )
```

Related Information

hanaclient_prepare Function [page 778]
hanaclient_execute_direct Function [page 763]

4.7.5.18 hanaclient_get_bind_param_info Function

Retrieves information about the parameters that were bound by using hanaclient_bind_param.

Syntax

```
hanaclient_get_bind_param_info ( <$stmt>, <$index> )
```

Parameters

$stmt	A statement successfully prepared by using hanaclient_prepare.
$index	The index of the parameter. The number must be between 0 and hanaclient_num_params() - 1.

Returns

Returns a two-element array where the first element is 1 on success or 0 on failure and the second element is a described parameter.

Example

```
# Get information on first parameter (0)
rc, param_info = api.hanaclient_get_bind_param_info( stmt, 0 )
print "Param_info direction = ", param_info.get_direction(), "\n"
print "Param_info output = ", param_info.get_output(), "\n"
```

Related Information

hanaclient_bind_param Function [page 755]
hanaclient_describe_bind_param Function [page 759]
hanaclient_prepare Function [page 778]

4.7.5.19 hanaclient_get_column Function

Returns the value fetched for the specified column.

Syntax

```
hanaclient_get_column ( <$stmt>, <$col_index> )
```

Parameters

$stmt	A statement object that was executed by hanaclient_execute or hanaclient_execute_direct.
$col_index	The number of the column to be retrieved between 0 and hanaclient_num_cols() - 1.

Returns

Returns a two-element array where the first element is 1 on success or 0 on failure and the second element is the column value.

Example

```
stmt = api.hanaclient_execute_direct( conn, "SELECT * FROM "Products"" )
# Fetch the second row
rc = api.hanaclient_fetch_next( stmt )
rc, employeeID = api.hanaclient_get_column( stmt, 0 )
rc, managerID  = api.hanaclient_get_column( stmt, 1 )
rc, surname = api.hanaclient_get_column( stmt, 2 )
rc, givenName = api.hanaclient_get_column( stmt, 3 )
rc, departmentID = api.hanaclient_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "\n"
```

Related Information

hanaclient_execute Function [page 762]
hanaclient_execute_direct Function [page 763]
hanaclient_fetch_absolute Function [page 765]
hanaclient_fetch_next Function [page 766]

4.7.5.20 hanaclient_get_column_info Function

Gets column information for the specified result set column.

Syntax

```
hanaclient_get_column_info ( <$stmt>, <$col_index> )
```

Parameters

$stmt	A statement object that was executed by hanaclient_execute or hanaclient_execute_direct.
$col_index	The column number between 0 and hanaclient_num_cols() - 1.

Returns

Returns a nine-element array of information describing a column in a result set. The array elements are described in the following table.

Element number	Type	Description
0	Integer	1 on success or 0 on failure.
1	Integer	Column index (0 to hana-client_num_cols() - 1).
2	String	Column name.
3	Integer	Column type.
4	Integer	Column native type.
5	Integer	Column precision (for numeric types).
6	Integer	Column scale (for numeric types).
7	Integer	Column size.
8	Integer	Column nullable (1=nullable, 0=not nullable).

Example

```
# Get column info for first column (0)
rc, col_num, col_name, col_type, col_native_type, col_precision, col_scale,
    col_size, col_nullable = api.hanaclient_get_column_info( stmt, 0 )
```

Related Information

hanaclient_execute Function [page 762]
hanaclient_execute_direct Function [page 763]
hanaclient_prepare Function [page 778]

4.7.5.21 hanaclient_get_next_result Function

Advances to the next result set in a multiple result set query.

Syntax

```
hanaclient_get_next_result ( <$stmt> )
```

Parameters

$stmt	A statement object executed by hanaclient_execute or hanaclient_execute_direct.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
stmt = api.hanaclient_prepare(conn, "call two_results()" )
rc = api.hanaclient_execute( stmt )
# Fetch from first result set
rc = api.hanaclient_fetch_absolute( stmt, 3 )
# Go to next result set
rc = api.hanaclient_get_next_result( stmt )
# Fetch from second result set
rc = api.hanaclient_fetch_absolute( stmt, 2 )
```

Related Information

hanaclient_execute Function [page 762]

hanaclient_execute_direct Function [page 763]

4.7.5.22 hanaclient_init Function

Initializes the interface.

Syntax

```
hanaclient_init ( )
```

Returns

Returns a two-element array where the first element is success or 0 on failure and the second element is the Ruby interface version.

Example

```
# Load the hanaclientwhere gem
begin
    require 'rubygems'
    gem 'hanaclientwhere'
    unless defined? hanaclientwhere
        require 'hanaclientwhere'
    end
end
# Create an interface
api = hanaclientwhere::hanaclientwhereInterface.new()
# Initialize the interface (loads the DLL/SO)
hanaclientwhere::API.hanaclient_initialize_interface( api )
# Initialize our api object
api.hanaclient_init()
```

Related Information

hanaclient_fini Function [page 767]

4.7.5.23 hanaclient_new_connection Function

Creates a connection object.

Syntax

```
hanaclient_new_connection ()
```

Returns

Returns a scalar value that is a connection object.

Remarks

A connection object must be created before a database connection is established. Errors can be retrieved from the connection object. Only one request can be processed on a connection at a time.

Example

```
require 'hanaclientwhere'
api = hanaclientwhere::hanaclientwhereInterface.new()
hanaclientwhere::API.hanaclient_initialize_interface( api )
api.hanaclient_init()
# Create a connection
conn = api.hanaclient_new_connection()
puts "Enter user ID"
myuid = STDIN.gets.chomp
puts "Enter password"
mypwd = STDIN.gets.chomp
# Establish a connection
status = api.hanaclient_connect( conn, "UID="+myuid+";PWD="+mypwd )
print "Connection status = #{status}\n"
```

Related Information

hanaclient_connect Function [page 758]
hanaclient_disconnect Function [page 761]

4.7.5.24 hanaclient_num_cols Function

Returns the number of columns in the result set.

Syntax

```
hanaclient_num_cols ( <$stmt> )
```

Parameters

$stmt	A statement object executed by hanaclient_execute or hanaclient_execute_direct.

Returns

Returns a scalar value that is the number of columns in the result set, or -1 on a failure.

Example

```
stmt = api.hanaclient_execute_direct( conn, "SELECT * FROM "Products"" )
# Get number of result set columns
num_cols = api.hanaclient_num_cols( stmt )
```

Related Information

hanaclient_execute Function [page 762]
hanaclient_execute_direct Function [page 763]
hanaclient_prepare Function [page 778]

4.7.5.25 hanaclient_num_params Function

Returns the expected number of parameters that are expected for a prepared statement.

Syntax

```
hanaclient_num_params ( <$stmt> )
```

Parameters

$stmt	A statement object returned by the successful execution of hanaclient_prepare.

Returns

Returns a scalar value that is the number of parameters in a prepared statement, or -1 on a failure.

Example

```
stmt = api.hanaclient_prepare(conn, "UPDATE "Products"
      SET "Products".ID = "Products".ID + 1000
      WHERE "Products".ID >= ?" )
num_params = api.hanaclient_num_params( stmt )
```

Related Information

hanaclient_prepare Function [page 778]

4.7.5.26 hanaclient_num_rows Function

Returns the number of rows in the result set.

Syntax

```
hanaclient_num_rows ( <$stmt> )
```

Parameters

$stmt	A statement object executed by hanaclient_execute or hanaclient_execute_direct.

Returns

Returns a scalar value that is the number of rows in the result set. If the number is an estimate, then it is negative and the estimate is the absolute value of the returned integer. The value returned is positive if the number of rows is exact.

Remarks

By default, this function only returns an estimate. To return an exact count, set the ROW_COUNTS option on the connection.

The number of rows in a result set can be returned only for the first result set in a statement that returns multiple result sets. If hanaclient_get_next_result moves to the next result set, then hanaclient_num_rows still returns the number of rows in the first result set.

Example

```
stmt = api.hanaclient_execute_direct( conn, "SELECT * FROM "Products"" )
# Get number of rows in result set
num_rows = api.hanaclient_num_rows( stmt )
```

Related Information

hanaclient_execute Function [page 762]
hanaclient_execute_direct Function [page 763]

4.7.5.27 hanaclient_prepare Function

Prepares the supplied SQL string.

Syntax

```
hanaclient_prepare ( <$conn>, <$sql> )
```

Parameters

$conn	A connection object with a connection established by using hanaclient_connect.
$sql	The SQL statement to be prepared.

Returns

Returns a scalar value that is the statement object, or nil on failure.

Remarks

The statement associated with the statement object is executed by hanaclient_execute. Use hanaclient_free_stmt to free the resources associated with the statement object.

Example

```
stmt = api.hanaclient_prepare(conn, "UPDATE "Products"
     SET "Products".ID = "Products".ID + 1000
     WHERE "Products".ID >= ?" )
rc, param = api.hanaclient_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.hanaclient_bind_param( stmt, 0, param )
rc = api.hanaclient_execute( stmt )
```

Related Information

hanaclient_execute Function [page 762]
hanaclient_free_stmt Function [page 769]
hanaclient_num_params Function [page 776]
hanaclient_describe_bind_param Function [page 759]
hanaclient_bind_param Function [page 755]

4.7.5.28 hanaclient_rollback Function

Rolls back the current transaction.

Syntax

```
hanaclient_rollback ( <$conn> )
```

Parameters

$conn	The connection object to perform the rollback operation on.

Returns

Returns a scalar value that is 1 on success, 0 on failure.

Example

```
rc = api.hanaclient_rollback( conn )
```

Related Information

hanaclient_commit Function [page 758]

4.7.5.29 hanaclient_sqlstate Function

Retrieves the current SQLSTATE.

Syntax

```
hanaclient_sqlstate ( <$conn> )
```

Parameters

$conn	A connection object returned from hanaclient_new_connection.

Returns

Returns a scalar value that is the current five-character SQLSTATE.

Example

```
sql_state = api.hanaclient_sqlstate( conn )
```

Related Information

hanaclient_error Function [page 761]

4.7.5.30 Column Types

The following Ruby class defines the column types returned by some Ruby extension functions.

```
class Types
    A_INVALID_TYPE = 0
    A_BINARY       = 1
    A_STRING       = 2
    A_DOUBLE       = 3
    A_VAL64        = 4
    A_UVAL64       = 5
    A_VAL32        = 6
    A_UVAL32       = 7
    A_VAL16        = 8
    A_UVAL16       = 9
    A_VAL8         = 10
    A_UVAL8        = 11
end
```

4.7.5.31 Native Column Types

The following table defines the native column types returned by some Ruby extension functions.

Native type value	Native type
0	DT_NULL
1	DT_TINYINT
2	DT_SMALLINT
3	DT_INT
4	DT_BIGINT
5	DT_DECIMAL
6	DT_REAL
7	DT_DOUBLE
8	DT_CHAR
9	DT_VARCHAR1
10	DT_NCHAR
11	DT_NVARCHAR
12	DT_BINARY
13	DT_VARBINARY
14	DT_DATE
15	DT_TIME
16	DT_TIMESTAMP
17	DT_TIME_TZ
18	DT_TIME_LTZ
19	DT_TIMESTAMP_TZ
20	DT_TIMESTAMP_LTZ
21	DT_INTERVAL_YM
22	DT_INTERVAL_DS
23	DT_ROWID
24	DT_UROWID
25	DT_CLOB
26	DT_NCLOB
27	DT_BLOB
28	DT_BOOLEAN
29	DT_STRING
30	DT_NSTRING

Native type value	Native type
31	DT_LOCATOR
32	DT_NLOCATOR
33	DT_BSTRING
34	DT_DECIMAL_DIGIT_ARRAY
35	DT_VARCHAR2
45	DT_TABLE
48	DT_ABAPSTREAM
49	DT_ABAPSTRUCT
50	DT_ARRAY
51	DT_TEXT
52	DT_SHORTTEXT
53	DT_BINTEXT
54	DT_ALPHANUM
61	DT_LONGDATE
62	DT_SECONDDATE
63	DT_DAYDATE
64	DT_SECONDTIME
70	DT_CLOCATOR
71	DT_BLOB_DISK_RESERVED
72	DT_CLOB_DISK_RESERVED
73	DT_NCLOB_DISK_RESERVE
74	DT_ST_GEOMETRY
75	DT_ST_POINT
76	DT_FIXED16
77	DT_ABAP_ITAB
78	DT_RECORD_ROW_STORE
79	DT_RECORD_COLUMN_STORE

4.8 SQLDBC Library

SQL Library and Database Connectivity (SQLDBC) is an application programming interface (API) that serves as the basis for several of the SAP HANA client interfaces.

SQLDBC is an internal interface exclusive to SAP and is not available for use directly by applications.

The following APIs are based on SQLDBC:

- ADO.NET
- Go (golang)
- Node.js
- ODBC
- Python
- Ruby

SAP HANA HDBSQL, a command-line application, is also based on SQLDBC. Other applications that use SQLDBC are SAP S/4 HANA and SAP R3.

File Names

SQLDBC is deployed as a shared library for direct use by the SAP HANA database server and in the SAP kernel for ABAP application servers via the Database Shared Library (DBSL). The file names are as follows:

- `libSQLDBCHDB.so` (Linux and AIX)
- `libSQLDBCHDB.dll` (Microsoft Windows)
- `libSQLDBCHDB.dylib` (macOS)

DBCAPI is a shared library used by the Node.js, Go (golang), and Ruby drivers and contains a static copy of SQLDBC. It does not depend on the SQLDBC shared library. The file names are as follows:

- `libdbcapiHDB.so` (Linux and AIX)
- `libdbcapiHDB.dll` (Microsoft Windows)
- `libdbcapiHDB.dylib` (macOS)

Upsert Spatial Data to SAP HANA

All SQLDBC-based SAP HANA client components, including the ODBC, Python, Node.js, golang, and Ruby drivers, as well as HDBSQL, support directly upserting spatial data in WKT format to SAP HANA without using the ST_GeomFromWKT conversion function in the UPSERT statement. The spatial data can be given in UTF8, CESU8, UCS2, or UCS4 format. However, in order to use this feature, the connection property `spatialTypes` must be set to 1 through the connection string for applications that uses the SAP HANA Node.js, Python, golong, or Ruby driver. For ODBC applications, setting `spatialTypes` to 1 is not required, but the application should not set this property to 0 through the connection string or through SQLSetConnectAttr with the attribute of SQL_SQLDBC_SPATIALTYPES.

Related Information

Installing the Database Shared Library (DBSL)
UPSERT Statement (Data Manipulation)

4.8.1 SQLDBC Tracing

SQLDBC tracing can help diagnose errors in applications that use any SQLDBC-based API.

SQLDBC tracing can be activated in several different ways:

- Tracing configured by hdbsqldbc_cons
- Tracing environment variables
- Connection-specific tracing to the console

Trace Using the hdbsqldbc_cons Tool

Use hdbsqldbc_cons to set tracing commands to initiate diagnostic tracing. Diagnostic tracing is useful if you encounter either of the following issues:

- You experience problems while using an interface or application that is based on the SAP HANA SQLDBC interface, such as ADO.NET, ABAP (DBSL), or hdbsql.
- You must provide an SAP HANA SQLDBC client trace (also known as a SQLDBC trace) of the application workflow that reproduces a specific problem.

> **Note**
>
> - Enabling SQLDBC tracing results in a significant reduction in SQLDBC performance. Additionally, enabling tracing can result in sensitive information (such as connection properties, SQL text, and result set values) being included in the trace file.
> - SQLDBC does not delete previous trace content when starting a new trace and archives the original file, if any, by appending ".archive" to the original name.
> - If you are running your SAP HANA client on a Microsoft Windows 32-bit system, then use hdbsqldbc_cons32.
> - Specifying the `CONFIG` keyword ensures that the settings are written to the persistent store of trace settings as opposed to written to shared memory (backed by a memory mapped file) when parsing the command line.

See hdbsqldbc_cons Trace Options [page 789] for more information.

Trace Using Environment Variables

If hdbsqldbc_cons is not installed, or if you are using a standalone driver such as Node.js, then there are two environment variables, both of which must be set, that can act as overrides for the location of the trace file and the level of tracing:

- Microsoft Windows

```
SET HDB_SQLDBC_TRACEFILE=<path-to-trace-file>
```

The trace file name can be a relative or an absolute path.

```
SET HDB_SQLDBC_TRACEOPTS=<trace-options>
```

UNIX/Bash

```
export HDB_SQLDBC_TRACEFILE=<path-to-trace-file>
```

```
export HDB_SQLDBC_TRACEOPTS=<trace-options>
```

SQLDBC inherently trusts all environment variables and must be sanitized if untrusted parties can set them. To mitigate the possibility that a malicious user could overwrite an operating system file, the following rules are applied:

- A timestamp is added to the name given in HDB_SQLDBC_TRACEFILE
- If the trace file already exists, it cannot be overwritten

To disable the effect of the environment variables, they must be unset before the client process is started.

See SQLDBC Trace Options [page 786] for more information.

Connection-Specific Tracing to the Console

There are two connection options for tracing to the console:

traceFile	Specifies the trace file name. Only the stdout and stderr values are supported.
traceOptions	Implicitly enables tracing for all SQLDBC-based client connections. See SQLDBC Trace Options [page 786] for more information.

Related Information

Troubleshooting: SQLDBC Tracing [page 794]
ODBC Tracing and Trace Options [page 656]
Troubleshooting: ODBC Tracing [page 662]
Node.js Tracing [page 533]

4.8.1.1 SQLDBC Trace Options

SQLDBC trace options are common trace options for environment variables, connection tracing to a console, the Node.js trace callback, and the ODBC trace callback.

Trace options are a comma-separated list of option names with optional values:

```
<option>[=<value>][,<option>[=<value>] ...]
```

For the SQL, DEBUG, and DISTRIBUTION trace options, the following severity level values are supported:

- NONE
- FATAL
- ERROR
- WARNING
- INFO
- DEBUG

Each higher level contains the information from the previous level (FATAL traces the least amount of content for the category while DEBUG traces all of the content for the category). If the option is specified without a severity level value, the default level will be DEBUG.

If a category is specified multiple times, the last category to be specified decides what is enabled.

List of SQLDBC Trace Options

The following table lists the SQLDBC trace options:

Option	Value	Default	Description
API	`<severity-level>`	NONE	Traces driver API calls.
CSE	ON | OFF	OFF	Traces client-side encryption.
DEBUG	`<severity-level>`	NONE	Traces the internal debug code in the client.
DISTRIBUTION	`<severity-level>`	NONE	Traces distributed connections. This option is useful for tracking "10108 Session has been reconnected" errors.
FLUSH	ON | OFF	OFF	Enables frequent flushing of the trace buffer to the trace file. If FLUSH is specified without a value, FLUSH will be turned on. **Note** FLUSH is ignored with ODBC callback tracing.

Option	Value	Default	Description
OutBufferSize	`<size>`	64K	Specifies the size of the buffer to use to accumulate trace messages for the callback. The minimum value is 4K, and setting it to zero will disable the feature. This option is only supported for the node trace callback.
PACKET	`<packet-options>`	OFF	Traces network command protocol packets. `<packet-options>` can be ON, OFF, UNLIMITED, or an integer part data size limit optionally followed by K, M, or G. For example, "PACKET=10K". A part data size limit of 0 enables packet summary tracing with a minimum of packet trace output.
SIZE	`<size-options>`	UNLIMITED	Controls the maximum size of the trace file output, before archiving. `<size-options>` can be OFF, UNLIMITED, or a file size limit. **Note** SIZE is ignored with ODBC callback tracing.
SQL	`<severity-level>`	NONE	Traces SQL statements, results, and parameters.
TIMING	ON | OFF	OFF	When enabled, traces the elapsed, client, network, and server time for each call into SQLDBC.

Example

The following example will turn on SQL tracing to the DEBUG level, DEBUG tracing to the INFO level, and limit the file size to 100M:

```
traceOptions="SQL,DEBUG=INFO,SIZE=100M"
```

The following example shows how to use environment variables to trace an application using the Microsoft Windows command prompt:

```
set HDB_SQLDBC_TRACEFILE="%TEMP%/sqldbctrace.txt"
set HDB_SQLDBC_TRACEOPTS="DEBUG,SQL"
...reproduce issue using the client...
set HDB_SQLDBC_TRACEFILE=
set HDB_SQLDBC_TRACEOPTS=
```

The following example shows how to use environment variables to trace an application using the UNIX/Bash (or other equivalent) shell:

```
export HDB_SQLDBC_TRACEFILE="/tmp/sqldbctrace.txt"
export HDB_SQLDBC_TRACEOPTS="DEBUG,SQL"
...reproduce issue using the client...
unset HDB_SQLDBC_TRACEFILE
unset HDB_SQLDBC_TRACEOPTS
```

4.8.1.2 hdbsqldbc_cons Trace Options

Use hdbsqldbc_cons to trace applications that are based on SQLDBC other than ODBC. For ODBC-based applications, use hdbodbc_cons.

Trace Options

The following table lists the available SQLDBC trace options:

> **Note**
>
> CONFIG commands affect the configuration only, but not running processes (until a TRACE REFRESH or any TRACE command without CONFIG including those with process id).

Option	Command Line Option	Description
Help	`hdbsqldbc_cons -h`	Displays the command line help.
Compact the trace	`hdbsqldbc_cons COMPACT`	Compacts the trace configuration store of SQLDBC-based clients. Use the COMPACT on clients where trace configuration updates are performed frequently, for example, development or support systems.

Option	Command Line Option	Description
Trace flush	`hdbsqldbc_cons [CONFIG] TRACE FLUSH ON | OFF`	Controls the use of the internal buffer (4 KB in size) for buffering trace content before writing to the trace file. This setting helps reduce I/O activity when tracing is enabled and improves performance.

Trace only on an error	`hdbsqldbc_cons [CONFIG] TRACE ONLY ON ERROR <error_code_1> [,<error_code_2>>,...<error_code_1000>] [BUFFER <buffer_size>] [COUNT <count>]`	Enables/disables content tracing if specified error codes occur a specified number of times. The maximum number of error codes that can be tracked is 1000, and at least 1 error code must be specified. When this tracing is enabled, and a tracked error occurs, `<buffer_size>`/2 bytes of tracing content before the error and `<buffer_size>`/2 bytes of tracing content after the error are traced. `<buffer_size><count>` are optional, where the defaults are 4 KB and 1, respectively.
		
		In order for error content tracing to work most effectively, category tracing must also be enabled, for example:
		
		`hdbsqldbc_cons trace debug on level fatal`
		
		`hdbsqldbc_cons trace debug`

Option	Command Line Option	Description
Trace category	`hdbsqldbc_cons [CONFIG] TRACE <category> ON [LEVEL <severity>] | OFF`	Enables/disables a tracing category for all active and new processes. `<severity>` specifies the type of content that should be traced from one of the supported levels: • FATAL • ERROR • WARNING • INFO • DEBUG Each higher level contains the information from the previous level (FATAL traces the least amount of content for the category while DEBUG traces all of the content for the category). The default level is DEBUG. The following `<category>` values are supported: **SQL** Traces SQL statements, results, and parameters. All `<severity>` levels are applicable to this category. **DEBUG** Traces the internal debug code in the client. All `<severity>` levels apply to this category. **PACKET** Traces network command protocol packets. `<severity>` is not used with this category, but a size can be used instead of the ON value. The size value can be UNLIMITED, or an integer part data size limit optionally followed by K, M, or G. For example, "hdbsqldbc_cons trace packet 10K". A part data size limit of 0 enables packet summary tracing with a minimum packet trace output.

Option	Command Line Option	Description
		DISTRIBUTION Traces distributed connections. This option is useful for tracking *10108 Session has been reconnected* errors. All `<severity>` levels are applicable to this category. **API** Traces driver API calls. This category is only supported for the Node.js driver. **TIMING** Traces the elapsed, client, network and server time for each call into SQLDBC. `<severity>` is not used with this category.
Show trace settings	`hdbsqldbc_cons SHOW ALL`	Displays the current trace settings.
Turn on a trace level for all processes	`hdbsqldbc_cons TRACE <level> ON`	Activates a tracing level for all active and new processes, including: **SQL** Traces SQL statements, results, and parameters. **DEBUG** Traces the internal debug code in the client. **PACKET** Traces network command protocol packets. **DISTRIBUTION** Traces distributed connections. This option is useful for tracking *10108 Session has been reconnected* errors. **API** Traces driver API calls. This category is only supported for the Node.js driver. **TIMING** Traces the elapsed, client, network and server time for each call into SQLDBC.
Turn on a trace level for new processes	`hdbsqldbc_cons CONFIG TRACE <level> ON`	Actives a tracing level for all new processes.

Option	Command Line Option	Description

Turn off a trace level for all processes	All processes:
	`hdbsqldbc_cons TRACE <level> OFF`
	New processes:
	`hdbsqldbc_cons CONFIG TRACE <level> OFF`
	All tracing levels:
	`hdbsqldbc_cons TRACE OFF`
		Deactivates a tracing level for all active and new processes.

Set trace file name	`hdbsqldbc_cons TRACE FILENAME /tmp/SQLDBC-%p.txt`	Sets a trace output file name for all active and new processes. If *%p* is specified in the file name, then it is replaced with the process ID of the traced application. For clarity, and to avoid issues where only one process can write to the file, include *%p* in the file name so that each process writes its own trace file. The special file names "stdout" and "stderr" are supported to send trace information to stdout and stderr, respectively.
Trace encrypted content	`hdbsqldbc_cons [CONFIG] TRACE CSE ON|OFF`	Turns tracing on or off for content that is either sent or retrieved from a client-side encrypted column.

Requirements

You must have an SAP HANA client installed on your computer.

The hdbsqldbc_cons utility is installed by the SAP HANA client installation. By default, it can be found in the following locations:

On Linux and UNIX platforms	`/usr/sap/hdbclient`
On Microsoft Windows platforms	`C:\Program Files\sap\hdbclient` (64-bit client) `C:\Program Files (x86)\sap\hdbclient` (32-bit client)

Providing a Size Limit for a Trace File

You can set a size limit for a trace file by specifying the following:

```
[CONFIG] TRACE SIZE <size>
```

<size> Specifies the size limit of the trace file, in bytes. At least 8192 bytes are required. You can also specify UNLIMITED, if no size limit is required.

The size can also be specified with a unit: `<size><unit>`, where `<unit>` = K (Kilobytes), M (Megabytes), or G (Gigabytes). When enabled, this causes wrapping to occur for the content. The file for which the limit is reached is renamed to `<path-to-trace-filename>.archive` and a new file is opened for subsequent trace content under `<path-to-trace-filename>`.

Example

The following example shows how to trace a connection using SQLDBC tracing:

```
hdbsqldbc_cons SHOW ALL
hdbsqldbc_cons CONFIG TRACE FILENAME /tmp/SQLDBC-%p.txt
hdbsqldbc_cons TRACE DISTRIBUTION ON
hdbsqldbc_cons TRACE SQL ON LEVEL INFO
… reproduce issue using the client …
hdbsqldbc_cons TRACE OFF
```

The following example shows how to set a limited trace file for specific errors with 8K of surrounding log information and timestamps, for a maximum of 3 occurrences:

```
hdbsqldbc_cons SHOW ALL
hdbsqldbc_cons CONFIG TRACE FILENAME /tmp/SQLDBC-%p.txt
hdbsqldbc_cons TRACE TIMING ON
hdbsqldbc_cons TRACE ONLY ON ERROR -10807,-10709,129,153,403 BUFFER 8192 COUNT 3
...
hdbsqldbc_cons TRACE OFF
```

4.8.1.3 Troubleshooting: SQLDBC Tracing

Identify issues affecting your SQLDBC trace.

If no trace file is written to disk, then verify whether any of the following issues are affecting your trace session:

- Ensure that you are tracing a SQLDBC-based application as hdbsqldbc_cons only applies to SQLDBC. To trace an ODBC-based application, use hdbodbc_cons.
- Ensure that you enabled tracing on the correct machine. If several application servers are being used in a landscape, then determine which application server the application is running on.
- Ensure that the user running hdbsqldbc_cons is the same user that runs the application being traced.
- Ensure that the trace tool has the same version as the driver. The hdbsqldbc_cons version should correspond to `libSQLDBCHDB.so`.

- Ensure that you specified a trace file location that is accessible from the application and in which the user is allowed to write files.
- Ensure that you look for the trace file in the correct output directory. The trace file name is relative, so the trace is written in the working directory of the application. If necessary, specify an absolute path.
- Ensure that you include **%p** in the trace file name. If you did not specify **%p** and more than one process is using the interface, then only the first process can write into the trace file and you receive a trace file that may not contain the correct information. The trace file path might be overloaded by ABAP profile parameters, in which case the trace files are located in the instance's work directory.

If you are working in Microsoft Windows, then verify whether any of the following issues are affecting your trace session:

- Ensure whether or not you need to use the -u `<user>` and option to specify the SAPservice `<SID>` username for the process.
- Ensure that the user running hdbsqldbc_cons with -u has administrator privileges. To change trace settings for other users, administrator privileges are required. Run hdbsqldbc_cons from a new command prompt using the *Run as administrator* option.

Related Information

ODBC Tracing and Trace Options [page 656]
Troubleshooting: ODBC Tracing [page 662]
SQLDBC Tracing [page 785]

4.8.2 SQLDBC Connection Properties

Use SQLDBC connection properties in conjunction with the relevant API or application to connect to an SAP HANA database.

> **Note**
>
> Do not set connection properties to an empty string. The client substitutes any empty string values with the property's default setting.

The following table lists SQLDBC connection properties (property names are case insensitive):

> **Note**
>
> A valid setting for a Boolean property can be: “1”, “TRUE”, “YES”, or “ON” for a TRUE setting or “0”, “FALSE”, “NO”, or “OFF” for a FALSE setting. The setting of a Boolean property is case-insensitive.

Property	Value	Default	Description
`abapVarCharMode`	Boolean	FALSE	Specifies that in ABAP VARCHAR mode, a single space and an empty string are considered the same.
`allowLocalCompress`	Boolean	TRUE	Determines whether or not connections from the same machine as the SAP HANA server are compressed. The `compress` parameter and/or the `indexserver.ini` > *session* > *compression* server configuration control whether or not network compression is enabled. **Note** If the connection uses a proxy, do not use `allowLocalCompress`=FALSE as the connection may not be compressed when required.
`allowReconnectOnSelect`	Boolean	OFF	Allows reconnects to happen for SELECT queries which fail during a receive. By default, this property is set to OFF, which allows for backward compatibility.

Property	Value	Default	Description
`bindAddress`	`<ip-address>`	NULL	Specifies that the provided address is used as the source address for the socket connection. By default, the address preferred by the OS is used. If setting `bindAddress` to a hostname, that hostname should only resolve to a single IP address. In the majority of network configurations, specifying `bindAddress` is not required. **Note** Only specify `bindAddress` if it is required to connect in your network configuration.
`chopBlanks`	Boolean	FALSE	Removes trailing blanks from character output.
`chopBlanksInput`	Boolean	FALSE	Removes trailing blanks from character input.
`communicationTimeout`	Timeout in milliseconds	0 (disabled)	Aborts communication attempts to the server after the specified timeout. If set to a non-zero value, the driver will wait for a maximum of the specified number of milliseconds for any request to the server to complete. If the time has expired and a response has not been received, the driver returns an error.

Property	Value	Default	Description
`compress`	Boolean	FALSE	Enables or disables network compression for the connection. The `indexserver.ini` > *session* > *compression* server configuration defaults to FALSE. Setting this parameter overrides the setting of the `indexserver.ini` > *session* > *compression* parameter. The `allowLocalCompress` property controls if clients running on the same machine as the server use network compression when compression is requested.
`connDownRollbackError`	Boolean	FALSE	Setting this property to TRUE specifies that if the database connection is down due to network or SAP HANA server problems, the client rolls back the last transaction and sends and error to the application. The application can resend the last transaction to the server after properly handling the error. The following conditions must be true: • The session is recoverable in the server • There is a written transaction (uncommitted operation) in the session • The client failed to send the last SQL statement, or the last statement was sent to the server successfully, but both auto-commit and DDL auto-commit in the client are off. By default, this property is set to FALSE.

Property	Value	Default	Description
`connectTimeout`	Timeout in milliseconds	0 (limited by the `nodeConnectTimeout` connection property)	Aborts connection attempts after the specified timeout. `connectTimeout` limits the overall time for the entire connection to try all server addresses specified. See `nodeConnectTimeout` for per-node timeout.
`currentSchema`	`<schema>`	CURRENTUSER	Sets the current schema, which is used for identifiers without a schema.
`cursorHoldabilityType`	NONE, COMMIT, ROLLBACK, COMMIT_AND_ROLLBACK	COMMIT	Specifies whether the cursor for a result set remains open after the transaction is committed or rolled back. If a cursor remains open, then the application is able to fetch more rows from it. NONE means the cursor is always closed after a commit or rollback, while COMMIT means the cursor remains valid after a commit, but not after rollback. Holding cursors, particularly hold over rollback, causes the SAP HANA server to use more resources. Support for holding a cursor over ROLLBACK or COMMIT_AND_ROLLBACK is available in SAP HANA servers beginning with SAP HANA 2.4.

Property	Value	Default	Description
`emptyTimestampIsNull`	Boolean	TRUE	When enabled, DATE, TIME, SECONDDATE, and TIMESTAMP values inserted as empty strings are returned as NULLs. When disabled, these values are returned as the following out-of-band values: • `DATE` - one day before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 00:00.000000000 UTC BC • `TIME` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `SECONDDATE` - one second before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.000000000 UTC BC • `TIMESTAMP` - one hundred nanoseconds before 0001-01-01 00:00:00.0000000 UTC AD, which is 0001-12-31 23:59:59.999999900 UTC BC

Property	Value	Default	Description
`ignoreTopology`	Boolean	FALSE	Controls how the client uses the topology returned from the server. TRUE indicates to use the topology unless port-forwarding is detected and FALSE indicates to always ignore the topology. It is an error to specify both ignoreTopology=TRUE and siteType=PRIMARY or SECONDARY.
`isolationLevel`	TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE	TRANSACTION_READ_COMMITTED	Sets the isolation level for the connection.
`isoTimestampOutput`	`<string>`	FALSE	Displays whether or not 'T' is used as the separator between the Date and Time parts of a timestamp retrieved as a string: TRUE/FALSE. If FALSE, then a space character is used as the separator.
`key`	`<username>`	Empty	Specifies the key for the hdbuserstore.
`maxLazyDroppedStatements`	`<number>`	50	Controls the maximum number of queued statements to be dropped. Setting this value to 0 disables the lazy drop feature and causes a request to be sent to the server each time a statement is dropped by the application.
`maxVersion`	`<string>`		Specifies the maximum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.

Property	Value	Default	Description
`minVersion`	`<string>`		Specifies the minimum TLS version the client is willing/able to connect with. Valid values include: TLSv1, TLSv1.1, TLSv1.2, TLSv1.3.
`nodeConnectTimeout`	Timeout in milliseconds	60000	Aborts the connection attempt after the specified maximum timeout. The connection attempt typically fails immediately if the server cannot be found. If connection attempts to a particular server address are slow to fail, then using this connection property reduces the time before a successful connection by minimizing the time waiting to connect to individual addresses. On a host with multiple resolved IP addresses, the network timeout for each attempt to connect to an IP address will be equal to the `nodeConnectTimeout` divided by the number of resolved IP addresses. If this option is set to zero, the `connectTimeout` setting will be used. If both of the `nodeConnectTimeout` and `connectTimeout` are set to zero, then SQLDBC will use the system TCP/IP socket connect timeout to connect to the IP address.

Property	Value	Default	Description
`packetCaching`	Boolean	TRUE	Caching packets reduces the need to continually allocate and deallocate memory for packets. However, this increases the peak memory usage of the application, and for how long it stays at or near that peak.
`packetSize`	`<number>` in bytes	1048576 (1 MB)	Sets the maximum size of a request packet sent from the client to the server, in bytes. In some cases, request packets can be larger (for example, if a SQL statement or parameter value is larger than the packetSize). The minimum is 1 MB.
`packetSizeLimit`	`<number>` in bytes	2147483647 (2 GB)	Sets the maximum size for a request or reply packet, in bytes. If SQLDBC needs to allocate a packet larger than the limit, then it returns an error. If a reply packet is too large, then the connection is dropped. The minimum value is the packetSize value; the maximum value is 2 GB.

Property	Value	Default	Description
`prefetch`	Boolean	TRUE	Controls result set prefetching. When enabled, result set prefetch requests the next block of rows of a result set while the application is still processing the current block of rows. When fetching large result sets, result set prefetch can improve performance; however, it can also result in the client using more memory. In some cases, the server JobExecutor thread could be blocked from sending the reply for an indefinite period of time until the client receives the prefetch reply. If the result set contains LOB columns, then result set prefetch is always disabled.
`{ proxyHostname | proxy_host }`	`<host-name>`		Specifies the host name of the proxy server. The default proxy server is SOCKS5. If `proxyHttp` is set to true, the proxy server is assumed to be an HTTP proxy.
`proxyHttp`	Boolean	FALSE	When set to TRUE, enables HTTP proxy authentication and implies that ENCRYPT=TRUE. This property is only applicable if `proxyHostname` is specified and `webSocketURL` is not specified.

Property	Value	Default	Description
`{ proxyPassword | proxy_password }`	`<password>`		Specifies the password for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the Cloud Connector location ID value. The value must be entered as a base64 encoded value with a "1." prepended to the string.
`{ proxyPort | proxy_port }`	`<port-number>`		Specifies the port of the proxy server. The default is 1080.
`{ proxyScpAccount | proxy_scp_account }`	SAP BTP, Cloud Foundry Environment: `<location-ID>` SAP BTP, Neo Environment: `<subaccount>.<location-ID>`		Provides SAP Cloud Connector routing information to the Cloud Connector SOCKS proxy. In an SAP BTP, Cloud Foundry Environment, provide an optional `<locationID>` in plaintext. Internally, the value is converted to base64 encoding. This value overrides the `proxyPassword` connection property. In an SAP BTP, Neo Environment, either `proxyScpAccount` or `proxyUsername` must be specified. `<subaccount>` is mandatory and `<locationID>` is optional. Both values are entered as plaintext strings and are internally converted to base64 encoding. This value overrides the `proxyUsername` connection property.

Property	Value	Default	Description
`{ proxyUserName | proxy_userid }`	`<user-name>`		Specifies the user name for Basic HTTP Authentication or METHOD 02 SOCKS authentication. When using an SAP HANA Cloud Connector in the SAP BTP, Cloud Foundry Environment, this field is used to provide the JSON Web Token (JWT) value. When using an SAP HANA Cloud Connector in the SAP BTP, Neo Environment, this field is used to provide the subaccount and location ID details. The values must be base64 encoded manually using the format `1.base64(subaccount).base64(locationID)`.
`reconnect`	Boolean	TRUE	Controls whether the system automatically reconnects to the database instance after a command timeout or when the connection is broken. Reconnecting restores the old state (for example, if no transaction was open).

Property	Value	Default	Description
`resolveHostName`	IPv4First IPv6First	IPv4First	Controls which addresses to use in a server address, especially if an address or hostname resolves to multiple IP addresses. In all cases, stop attempting further resolved addresses once a TCP/IP connection is successfully established. The following values can be used with `resolveHostName`: • IPv4First (default) - try all resolved addresses starting with IPv4 addresses. • IPv6First - try all resolved addresses starting with IPv6 addresses.
`sessionVariable:<key-name>`	`<session variable value>`	None except for APPLICATION and APPLICATIONUSER keys	Sets the session variable `<key-name>` to `<session variable value>`. For example, to set the session variable with key `APPLICATION`, use `sessionVariable:APPLICATION=MyApplication`.

Property	Value	Default	Description
`statementCacheSize`	`<number>`	0 (no statements cached)	Controls the maximum number of cached prepared statements. Caching SQL statements that are frequently prepared and dropped can improve performance. DDL can cause behavior changes when statement caching is enabled. **Note** If it is possible that the definition of objects referred to by the prepared statements may change because DDL is executed while you are connected, then it is recommended that you do not enable prepared statement caching.
`statementRoutingFailureBackoff`	Boolean	TRUE	Enables or disables the re-establishment of an internal connection to a specific host for a time period, after the internal connection fails during the preparation or execution of a statement routed to that host. The minimum time period is 1 minute. The time period is doubled after each subsequent internal connection establishment or usage failure for the same node, up to a maximum of 1 hour. Every successful internal connection re-establishment and usage resets the time period to the minimum value.

Property	Value	Default	Description
`statementRoutingWarnings`	Boolean	FALSE	Returns a SQL warning to the application with SQLCODE -11001 or -11003 that gives information about the routing failure of a prepared statement.
`tcpKeepAliveCount`	`<number-of-retries>` This value is ignored if it is 0.	5	Controls the number of KeepAlive packets sent to the server before the driver assumes that the connection is closed. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.
`tcpKeepAliveIdle`	`<seconds>` This value is ignored if it is 0.	200	Controls the number of seconds that the connection is idle before the driver sends a KeepAlive packet. If 0 is specified, this value is determined by the operating system.
`tcpKeepAliveInterval`	`<seconds>` This value is ignored if it is 0.	4	Controls the number of specified seconds before the driver sends an additional KeepAlive packet if there was no response to the previous KeepAlive packet. On Windows and MacOS, or if 0 is specified, this value is determined by the operating system and is ignored on Microsoft Windows.

Property	Value	Default	Description
`timestampPadding`	Boolean	TRUE	In a timestamp where the format contains a fraction, this property determines whether the fractional portion of the timestamp will be 7 or 9 digits. When disabled, the fractional portion is 7 digits to match how the server displays timestamps. Otherwise, it is 9 digits. **Note** ABAP timestamps are not affected regardless of the value of `timestampPadding`.
`traceFile`			Specifies the trace file name. Only the stdout and stderr values are supported.
`traceOptions`			Implicitly enables tracing for all SQLDBC-based client connections. See the HDB_SQLDBC_TRACEOPTS description in the **Environment Variables** topic for more information.
`webSocketURL` (SAP HANA Service only)	`<webSocketUrl>`		Specifies the WebSocket URL used to connect to a SAP HANA server. The connection property value cannot be empty. WebSockets connections have the following default connection property settings: • encrypt=TRUE • compress=TRUE • reconnect=FALSE

Security Connection Properties

Property	Value	Default	Description
`{ cseKeyStorePassword | clientside_encryption_keystore_password }`	`<string>`		Provides the password for the local key store. This option is required when using client-side encryption. Passwords must be at least eight characters and have at least one upper case character, one lower case character, and one number.
`authenticationMethods`	`<comma-separated-string>`		Provides a comma-separated list of authentication methods to be used by the client. Possible values are: password, pbkdf2, ldap, saml, jwt, saplogon, kerberos, sessioncookie, and X509.
`authenticationX509`	`<string>`	`<value-of-sslKeyStore>`	Provides a string containing the X509 certificate or the name of a file containing the X509 certificate. The certificate must be PEM-encoded in PKCS8 format. Defaults to the existing value of `sslKeyStore`, if specified.
`authenticationX509Password`	`<password>`		With CommonCryptoLib, specifies the PIN for the PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`encrypt`	Boolean	FALSE	Enables or disables TLS encryption. For SAP HANA versions 2.5 and lower, set encrypt=TRUE when connecting to port 443 (SAP HANA Cloud). For SAP HANA versions 2.6 and higher, encrypt=TRUE is not required when connecting to port 443.

Property	Value	Default	Description
`sslCryptoProvider`	{ commoncrypto | openssl | mscrypto }	1. commoncrypto (if installed) 2. openssl/mscrypto	Specifies the cryptographic library provider used for TLS communication. If you specify a value for this property, then you must also explicitly specify paths in both the `sslKeyStore` and `sslTrustStore` properties to avoid configuration issues. If CommonCryptoLib is not available, OpenSSL is used by default in Linux environments, and msCrypto in Microsoft Windows environments. OpenSSL is not available on Microsoft Windows. **Note** Check the `client` folder of the installation package to see if `COMMONCRYPTOLIB.TGZ` is present. If not, you can download CommonCryptoLib separately. For instructions on how to download CommonCryptoLib, see "Download and Install SAP Common Crypto Library" in the *SAP HANA Client Installation and Update Guide*.
`sslHostNameInCertificate`	`<string>`	Empty	Specifies the host name used to verify server's identity. The host name specified here verifies the identity of the server instead of the host name with which the connection was established. For example, in a single-host system, if a connection is established from a client on the same host as the server, then a mismatch would arise between the host named in the certificate (actual host name) and the host used to establish the connection (localhost). If you specify `*` as the host name, then the server's host name is not validated. Other wildcards are not permitted.

Property	Value	Default	Description
`sslKeyStore`	`<file>` | `<PEM-encoded-identity>` msCrypto: MY | Root | Trust | CA	`$SECUDIR/sapcli.pse` (CommonCryptoLib) `$HOME/.ssl/key.pem` (OpenSSL) MY (msCrypto)	Specifies the path to the keystore file that contains the client's identity. An identity is used for mutual TLS authentication and consists of the client's private key, the client's certificate, and, optionally, the certificate of the signing authority that signed the client's certificate. If you are using CommonCryptoLib, then the PSE file must also contain the trust store (for example, the server's public certificates) and the sslTrustStore property should be empty. If you are using CommonCryptoLib, but not doing mutual TLS authentication, then the PSE file contains only the trust store. If you are using CommonCryptoLib, then use the SAPGENPSE tool (installed with CommonCryptoLib) to create the client PSE (`sapcli.pse`) and generate the client's keys. You must also import the server's public certificates into `sapcli.pse`. Typically, this is the root certificate or the certificate of the certification authority that signed the server's public certificates. See *SAP Note 1718944 (SAP HANA DB: Securing External SQL Communication (CommonCryptoLib))*. If you are using OpenSSL, use OpenSSL tools to create the required keystore file. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.

Property	Value	Default	Description
`sslMinProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	TLS12	Controls the minimum TLS version. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The lowest protocol version is accepted in TLS handshakes.
`sslMaxProtocolVersion`	TLS10, TLS11, TLS12, TLS13, and MAX	MAX	Controls the maximum TLS version. MAX allows this parameter to be future-proof in the event that new TLS versions are supported. Applies only to the CommonCryptoLib and OpenSSL Crypto Providers. The highest negotiated protocol version is used in TLS handshakes.
`sslKeyStorePassword`	`<password>`		With CommonCryptoLib, specifies the PIN for the keystore PSE or the password for the encrypted key. With OpenSSL, specifies the password for the encrypted key.
`sslSNIHostname`	`<string>`	Empty	Specifies the name of the host that is attempting to connect at the start of the TLS handshaking process.
`sslSNIRequest`	Boolean	TRUE	Specifies whether SNI requests are enabled for TLS connections: TRUE/FALSE.

Property	Value	Default	Description
`sslTrustStore`	`<file>` | `<PEM-encoded-trust-store>` msCrypto: MY | Root | Trust | CA	`$HOME/.ssl/trust.pem` (OpenSSL) sapcli.pse (CommonCryptoLib) MY (msCrypto)	Specifies the path to a trust store file that contains the server's public certificates if using OpenSSL. Typically, the trust store contains the root certificate or the certificate of the certification authority that signed the server's public certificates. If you are using the cryptographic library CommonCryptoLib or msCrypto, leave this property empty. Alternatively, the PEM-encoded client identity can be provided directly as a string. Use the SAPGENPSE tool to export the contents of a PSE file in PEM format. When you provide the client identity as a string, use the sslTrustStore property to provide the PEM-encoded trust store as a string.
`sslUseDefaultTrustStore`	Boolean	TRUE	(Only applies to the OpenSSL Crypto Provider) Set to FALSE in order to disable the use of the default Certificate Authorities provided with OpenSSL or the Operating System.
`sslValidateCertificate`	Boolean	TRUE	Specifies whether to validate the server's certificate.

Related Information

ADO.NET Connection Properties [page 41]
Connect Method and Python Connection Properties [page 668]
Go Connection Properties [page 407]
Node.js Connection Properties [page 506]
ODBC Connection Properties [page 621]
Ruby Connection Properties [page 737]
SAP HANA HDBSQL Options [page 858]
Environment Variables [page 20]

5 Client Support for Active/Active (Read Enabled) (SAP HANA Platform)

Active/Active (read enabled) enables SAP HANA system replication to support read-only access to the secondary system and is supported by all client APIs.

When using Active/Active (read enabled), clients can connect directly to either the primary or secondary system.

5.1 Connecting Using Active/Active (Read Enabled) (SAP HANA Platform)

Connecting to an SAP HANA server by using Active/Active (read enabled) enables you to take advantage of a secondary system for better overall performance.

There are two ways to access the read-enabled secondary system:

Explicit read-only connection	The application opens an explicit connection to the secondary system.
Hint-based statement routing	The application connects directly to the primary system. On this connection, SQL statements with system replication-specific hints are prepared and executed. They are then automatically routed to and executed by the secondary system. For more information about hints, see the *SAP HANA SQL and System Views Reference Guide*.

If multiple addresses are specified for a connection, then all addresses must be for SAP HANA server nodes in either the primary or secondary system, but not both. Providing addresses for multiple systems is not allowed and may result in an error. However, if you specify the siteType connection property, then you can specify addresses for both the primary and secondary systems.

Related Information

Hint-Based Statement Routing for Active/Active (Read Enabled) (SAP HANA Platform) [page 817]
Forced Statement Routing to a Site for Active/Active (Read Enabled) (SAP HANA Platform) [page 817]
Implement Forced Statement Routing to a Site for Active/Active (Read Enabled) Using ODBC (SAP HANA Platform)

5.2 Forced Statement Routing to a Site for Active/Active (Read Enabled) (SAP HANA Platform)

Applications connecting to an Active/Active (read enabled) system can force the execution of a specific statement to the secondary site to move server processing to the secondary system. This reduces the load on the primary system and increases overall performance.

Forced statement routing can only be used when the following criteria are met:

- The system is using Active/Active (read enabled)
- The secondary site is accessible
- The connection's transaction isolation level is not set to REPEATABLE or SERIALIZABLE
- The connection does not have a write transaction in progress (an uncommitted INSERT, UPDATE, or DELETE)
- The initial connection that was made when the application established the connection is physically connected to the primary site

You cannot force route from a secondary site to a primary site.

Related Information

Implement Forced Statement Routing to a Site for Active/Active (Read Enabled) Using ODBC (SAP HANA Platform)
Hint-Based Statement Routing for Active/Active (Read Enabled) (SAP HANA Platform) [page 817]
Connection Types
Performance: Using Hints to Query Data Snapshots

5.3 Hint-Based Statement Routing for Active/Active (Read Enabled) (SAP HANA Platform)

Connections to a primary system can use hint-based statement routing statement execution to a secondary system on a per-statement basis. This reduces the load on the primary system and increases overall performance.

To indicate that a statement should be hint-based routed to the secondary system, add the hint text `WITH HINT(RESULT_LAG('hana_sr'))` to the end of the SQL SELECT statement. For example:

```
SELECT C1, C2 FROM T1 WHERE C3 = 'constant value' WITH
HINT(RESULT_LAG('hana_sr'))
```

Queries that are executed directly (not prepared) are not hint-based routed even if they contain a hint. To take advantage of hint-based statement routing, there must be separate prepare and execute operations at the SQLDBC or JDBC level. In some cases, applications or interfaces that use SQLDBC or JDBC (such as SAP HANA Cockpit, ABAP, or PyDBAPI) can perform a separate prepare and execute without the user's knowledge.

Routing Conditions

In the following cases, the execution transparently falls back to the primary system, even if the statement contains a hint to route to the secondary system:

- The connection's isolation level is set to repeatable read or serializable.
- Connection to the secondary system is not possible (for example, there is a secondary system outage, a networking issue, and so on).
- The connection currently has a write transaction (uncommitted insert, update, or delete) in progress.
- The query references temporary tables.

If the routed connection is dropped while fetching from a hint-routed query result set, an error may be returned to the application.

> **Note**
>
> Hint-based statement routing is only applied to SELECT statements.

Fallback Routing

In the following cases, a statement that has been routed to the secondary system gets re-routed to the primary system:

- The hint contains a maximum delay time parameter and the secondary system is delayed by more than that amount.
- The secondary system is near its maximum memory usage.

Timeout

If the previous hint-based routed statement execution falls back to the primary system due to a connection or communication error, then future hint-based statement routing does not attempt to re-connect to the secondary system for several seconds. This avoids the performance cost of retrying the connection to the secondary system frequently when it is likely to fail.

In this case, the time between reconnection attempts to the secondary system is between five seconds and five minutes from the last reconnection attempt. The time between reconnection attempts automatically increases if reconnection attempts continue to fail.

In a multitier system replication system, hint-based statement routing always routes from the primary to the secondary system.

Related Information

HINT Details

Connection Types

6 Application Tuning

Tuning an application can have a greater performance impact than tuning SAP HANA.

A number of SAP HANA client features can be enabled or changed to improve the behavior of an application that is connected to an SAP HANA database.

6.1 Tuning an Application

Tuning your application can reduce both the number of requests and the amount of data transferred.

The following recommendations can improve application performance in any environment, such as the client and server on the computer or client and server connected by a LAN (local area network) or a WAN (wide area network):

- Move logic that requires many SQL statements from the application to one or more stored procedures or functions.
- Avoid changing the current schema after initializing the connection. Changing the schema prevents prepared statement caching from reusing any statements that were cached before the schema was changed.
- Using DDL in a production system can result in potential performance issues and is often a sign of poor application design. DDL on temporary (non-persistent) objects is an exception and is less likely to impact performance.
- Always use prepared statements for multi-node SAP HANA systems. Even if the statement is only executed once, the client can take advantage of statement routing to execute statements more efficiently across multiple SAP HANA nodes.
- Ensure that your application is not executing queries or SQL statements more often than is necessary. If a particular query is executed more than once, then consider changing the application to cache the results the first time that the query is executed and then use the cached values rather than re-executing the query.
- Combine queries that get one function or expression value into one multiple-column query. For example, if you want to execute the following queries:

  ```
  SELECT CURRENT_SCHEMA FROM DUMMY
  ```

  ```
  SELECT CURRENT_USER FROM DUMMY
  ```

  ```
  SELECT CURRENT_CONNECTION FROM DUMMY
  ```

  Instead of executing three separate queries, use a single query, such as:

  ```
  SELECT CURRENT_SCHEMA, CURRENT_USER, CURRENT_CONNECTION FROM DUMMY
  ```
- Avoid creating joins in the application with multiple queries if it is possible to perform the join in the server. If an application executes a query and then executes a second query using the result from the first query,

then you may be joining multiple queries together in your application. If it is possible to perform the join by using one query instead of many, then this change can significantly improve performance.

- Avoid many short-running connections when you can use a single long running connection instead.
- Instead of getting each result set column value separately, bind data values to columns (for interfaces that support this). For example, in ODBC use SQLBindCol instead of SQLGetData.
  When you are getting result set values, consider getting a block of row values with one fetch call (for interfaces that support this). For example, in ODBC set the SQL_ATTR_ROW_ARRAY_SIZE statement attribute.
- Use the server data types BINARY(<n>) or NVARCHAR(<n>) instead of BLOB or NCLOB when possible (for instance when a maximum <n> of 5000 is large enough for any data that you want to store in the column or value). The SAP HANA client, server, and protocol can more efficiently process BINARY(<n>) or NVARCHAR(<n>) values compared to BLOB or NCLOB values.

Related Information

Reducing the Amount of Data Transferred on the Network [page 830]
Reducing the Number of Requests [page 831]
Best Practices for Node.js [page 831]
Improve Node.js Performance with Connection Pooling [page 832]
Binary Data Types
Character String Data Types
Large Object (LOB) Data Types

6.2 Performance Tuning

6.2.1 Packet Size

SAP HANA clients and the SAP HANA server encode data into packets for transfer between the clients and server.

A call into a client API method or function may require a different number of round trips depending on the packet size. For example, processing batch executes, fetching result sets, and transferring LOB data can require more than one round trip depending on the size of the data and the packet size. Increasing the packet size can reduce the number of round trips for these operations and can improve performance.

> **Note**
>
> Increasing the packet size can either improve or reduce performance, depending on the situation. For example, increasing the packet size increases the amount of memory that the client and the server use, which can reduce the responsiveness of certain types of requests. Therefore, implement performance testing when adjusting the packet size.

You can change the packet size at either the connection level or the statement level. The client uses the default connection level packet size of 1 MB for communication with the server. Changing the connection level packet

size will apply to all statements created with the connection, but changing the statement packet size only applies to the statement. For example, if the statement needs to send a large amount of data to the server, increasing the statement packet size may increase the performance without impacting any other statements or needing to establish a connection with a different packet size.

Set the Packet Size for a Connection

Setting the packet size at the connection level has the same effect on all of its statements.

Set the Connection `packetSize`

Set the packet size for a connection using the `packetSize` connection property. See SQLDBC Connection Properties [page 795] and JDBC Connection Properties [page 441] for more information.

Set the Packet Size for a Connection in ODBC

As an alternative to the `packetSize` connection property, the following ODBC API can be used to set the connection packet size:

```
SQLSetConnAttr(hdbc, SQL_ATTR_PACKET_SIZE, (SQLPOINTER)packetSize,
SQL_IS_INTEGER);
```

> ⚠ Restriction
>
> This API call must be made before invoking the SQLDriverConnect or SQLConnect API or it will be ignored.

Set the Packet Size for a Statement

You can change the packet size for each statement created from a connection. The packet size specified for a statement overrides the connection level packet size for that statement and only affects that statement.

When setting the statement packet size, the value provided for `packetSize` must be greater than the default packet size for the connection or the connection's `packetSize` property (if specified), otherwise the default packet size or the specified `packetSize` is used.

Set the Packet Size for an ODBC Statement

Applications that use the ODBC driver can change the packet size for the statement using the following API:

```
SQLSetStmtAttr(hstmt, SQL_ATTR_STMT_PACKET_SIZE, (SQLPOINTER)packetSize,
SQL_IS_INTEGER);
```

Set the Packet Size for a JDBC Statement

Applications that use the JDBC driver can use the following extension functions to set or get a packet size per statement:

- ```
  com.sap.db.jdbc.Driver.setPacketSize( java.sql.Statement statement, int
  packetSize )
  ```
  ```

- `com.sap.db.jdbc.Driver.getPacketSize( java.sql.Statement statement )`

Set the Packet Size for a Python Statement

Applications that use the Python driver can use the following extension functions to set or get a packet size per cursor:

- `cursor.setpacketsize(packetSize)`
- `cursor.getpacketsize()`

Related Information

ODBC Statement Attributes [page 645]
JDBC Driver Static Methods [page 476]
Cursor Class [page 698]

6.2.2 Prepared Statement Caching

Caches frequently prepared SQL statements.

Prepared statement caching reduces the number of requests to the SAP HANA server when the same SQL statement is prepared, dropped, or closed by an application multiple times. The SAP HANA client caches the statement, leaving it prepared on the database server even after it has been dropped by the application. Consider an application that repeats the following operations:

```
PREPARE, EXECUTE, DROP, PREPARE, EXECUTE, DROP, PREPARE, EXECUTE, DROP...
```

When prepared statement caching is enabled, the client can optimize this sequence into fewer requests to the server:

```
PREPARE, EXECUTE, EXECUTE, PREPARE, EXECUTE, EXECUTE...
```

Reducing the number of server requests can provide a significant performance improvement.

The SAP HANA client caches statements with SQL strings that have been prepared on the same connection. Once the cache is full, statements that are considered the most likely to be reused (based on the number of times that they have been prepared and how recently they were used) are preferred.

Prepared Statements

Applications that use separate Prepare and Execute operations can benefit from prepared statement caching. Direct execute operations (those with no separate prepare operation) do not receive any benefit from prepared statement caching. Examples of prepare operations for a number of SAP HANA client interfaces:

API	Prepare Operation
Microsoft ADO.NET	HanaCommand constructors that take a SQL string
Go	func (c *Conn) PrepareContext
JDBC	Connection prepareStatement and prepareCall methods
ODBC	SQLPrepare function
Node.js	connection.prepare methods
Ruby	hanaclient_prepare function

Choosing a Maximum Statement Cache Size

Prepared statement caching is disabled by default and is enabled by specifying the maximum statement cache size with the `statementCacheSize` connection property.

Choosing a reasonable maximum statement cache size is application dependent. Some general guidelines are:

- A good cache size maximizes the performance benefit from prepared statement caching.
- A maximum cache size of between 20 and 200 is appropriate for many applications.
- If you know a connection is likely to prepare and drop a certain number of unique SQL statements at least three times each, then the number of unique SQL statements is likely an upper bound for the maximum cache size.
- Both the SAP HANA client and the SAP HANA server use more memory as the cache size increases.
- If you look at the statement caching statistics traced by your application and the `PreparedStatementCacheRejectedFullCount` is at least half of the `PreparedStatementCacheHitCount`, then increasing the maximum cache size can improve performance.
- Applications that do not drop and re-prepare the same unique SQL statement text do not benefit from prepared statement caching.

View Prepared Statement Caching Effectiveness

For every disconnect, the distribution trace and JDBC STATISTICS trace logs information regarding the effectiveness that prepared statement caching is having on the system (if enabled). The following values provide insight into the effectiveness of prepared statement caching:

PreparedStatementCount

States the number of prepares performed by the connection.

PreparedStatementCacheHitCount

Indicates the number of times prepared statement caching improved performance. States the number of application prepares that reused a cached statement instead of preparing the same statements multiple times on the server.

PreparedStatementCacheRejectedFullCount

States the number of times that the application dropped a statement that was eligible to be cached, but was not, because the cache was already full of statements considered more likely to be reused. If this value is high relative to PreparedStatementCacheHitCount, then increasing the maximum statement cache size is likely to improve performance.

PreparedStatementCacheEvictedFullCount

States the number of times the application dropped a statement that was cached, but to cache that statement, another cached statement considered less likely to be reused was dropped. If this value is high relative to PreparedStatementCacheHitCount, then increasing the cache size may improve performance. If the workload of the connection changes over time, then this value may be high but increasing the cache size may not improve performance.

Other values related to prepared statement caching may also be logged in trace files but they are for SAP internal use only.

Affected Behaviors

If prepared statement caching is enabled, then the following scenarios show how it can affect application behavior. If you enable prepared statement caching and perform DDL while your application is running and statements are cached, then your application could suffer from similar behavior changes.

Scenario 1: Stale Describe Information

1. The SAP HANA client caches a prepared statement (for example, `SELECT * FROM T1`).
2. The application executes DDL, likely on a different connection, which changes the description (the column names or types) of the prepared statement's result set or parameters.
3. The application prepares the statement again, reusing the cached statement.
4. The application describes the statement (for example, with the ODBC SQLDescribeCol function or the JDBC ResultSet getMetaData method).

Because prepared statement caching is enabled, the application gets stale describe information. Without the cached statement, the described information would have been correct.

Scenario 2: Stale Parameter Information

1. The SAP HANA client caches a prepared statement, for example `INSERT INTO T1 VALUES(?)` where T1 has a single INTEGER column.
2. The application executes DDL so that T1's column is altered to a DECIMAL type.
3. The application prepares the statement again, reusing the cached statement.
4. The application executes the statement with a bound decimal parameter value of 1.2345.

Because prepared statement caching is enabled and client-side parameter conversion is used, the client converts the parameter to an INTEGER, which loses the fraction value (so 1 is inserted instead of 1.2345).

Without prepared statement caching, the full value is sent to the server. Depending on the initial and the altered types, similar sequences could result in a conversion error when the statement is executed.

6.2.3 Result Set Prefetch

Improve performance when fetching large result sets.

Result set prefetch is enabled by default. It improves performance even when the client is connected to older SAP HANA servers. It is disabled if a result set contains LOB columns.

Example

An application executes a query that returns 80,000 rows. By default, the first 32 rows are returned in the initial reply, the next 32768 rows (rows 33-32800) are returned in the second reply, the following 32768 rows (rows 32801-65568) are returned on the third reply, and so on. With prefetch enabled the driver sends the request for rows 32801-65568 when the application moves to the middle row of the previous request (around row 16415) rather than waiting until row 32801 is required. The goal is to ensure that rows 32801 is already fetched when required.

6.2.4 Control maxLazyDroppedStatements

Control the maximum number of queued statements to be dropped lazily. The drop statement requests are piggybacked on the next non-drop statement request sent by the client.

The default is 50 rows, which can significantly reduce the number of requests sent by the client but can also increase the number of open statements on the server.

Setting to 0 disables the lazy drop feature which causes a request to be sent to the server each time the statement is closed by the client; for example, in JDBC when PreparedStatement.close() is invoked.

6.3 Data Transfer Tuning

6.3.1 Network Compression

Network compression improves performance by compressing packets sent between the client and server.

When to enable network compression

Network compression almost always improves performance when the SAP HANA client and server are in different data centers, including accessing an SAP HANA Cloud. Network compression also often improves performance for same data center connections. Consider running some performance analysis on your same data center connections before determining whether enabling network compression is beneficial.

In the following cases, network compression can potentially decrease performance::

CPU overhead	Network compression increases CPU and memory usage on the client and server. If the SAP HANA server is already using near the maximum CPU usage on all cores, then enabling network compression can potentially decrease performance.
Data that does not compress well	If most of the data that is transferred between the client and server is not compressible (for example, result set values and parameter values that are already compressed or encrypted), then enabling network compression can potentially decrease performance.
Network speed	If the network is already extremely fast or the client and server are on the same computer, then enabling network compression can potentially decrease performance.

Network compression of packets occurs before network encryption, so enabling network compression when you are using network encryption can improve performance.

6.3.1.1 Configure Network Compression

Enable or disable network compression for all client connections or for specific connections.

Context

Network compression can improve performance by compressing packets that are sent between the client and server.

By default, network compression is always enabled for SAP HANA Cloud. For SAP HANA Platform and SAP HANA Service network compression is disabled.

Procedure

To enable network compression for a connection:

- Connections to SAP HANA Cloud are always compressed as of QRC 2/2023. The client `compress` connection property set to FALSE can be used to disable compression. The client `compress` connection property set to TRUE enables compression for earlier releases of SAP HANA Cloud.
- For SAP HANA Platform and SAP HANA Service, compression is disabled by default. To enable compression, either or both of the following options are set as follows:
  - The `indexserver.ini` > *session* > *compression* configuration parameter is set to TRUE and the client `compress` connection property is either TRUE or not specified.
    Changing the `indexserver.ini` > *session* > *compression* configuration parameter to TRUE is recommended when you want all (or most) connections to use network compression.
  - The client compress connection property is set to TRUE (regardless of the `indexserver.ini` > *session* > *compression* setting).
    Set the client compress connection on a per-connection basis to override the `indexserver.ini` > *session* > *compression* default.
- For SAP HANA Platform, local same machine connections are also compressed if compression is enabled. To disable local compression, set the `allowLocalCompress` connection property to FALSE.

> **Note**
>
> Compressing the same machine connections is likely to cause reduced performance.

Results

You have configured network compression for either all client connections or a specific client connection.

Related Information

SAP Note 2036111
JDBC Connection Properties [page 441]

6.3.1.2 Determine Network Compression Rate

Use distribution or statistic tracing to determine whether network compression is enabled and to determine the compression rate of a connection.

Context

Distribution or statistics tracing does not noticeably impact performance (unlike other tracing levels).

Procedure

1. Choose one of the following options, depending on your client interface:

Option	Description
Enable distribution tracing for ODBC applications	Use the command `hdbodbc_cons TRACE distribution ON`
Enable distribution tracing for SQLDBC-based APIs	Use the command `hdbsqldbc_cons TRACE distribution ON`
Enable statistics tracing for JDBC applications	Use the command `java -jar <installation_path>\ngdbc.jar TRACE STATISTICS ON`

2. Run your application with network compression enabled.
3. Disconnect your application. The compression rate information is added to your trace file.
4. View your trace file.

   For example, for every disconnect, the trace file contains at least one section of lines with a format similar to the following:

```
::PHYSICAL CONNECTION CLOSE 2017-05-18 12:23:21.009000 [0x000002b206694a30]
NUM RECV ON PCONN: 52, SUM RECV TIME: 761 MSEC
DATA RECV ON PCONN: 15.21 MB UNCOMPRESSED: 49.02 MB (COMPRESSION RATE: 3.22)
NUM SEND ON PCONN: 52, SUM SEND TIME: 1 MSEC
DATA SENT ON PCONN: 0.02 MB UNCOMPRESSED: 0.02 MB (COMPRESSION RATE: 1.00)
::CLOSE CONNECTION END 2017-05-18 12:23:21.009000 [0x000002b20668cf30]
```

   These lines show the amount of data actually sent and received, the amount of data that the client would have sent and received if network compression were disabled (the UNCOMPRESSED values), and the compression rates, which are calculated as the uncompressed size divided by the compressed size.

   The UNCOMPRESSED and COMPRESSION RATE lines only appear in your trace file if network compression is enabled for your connection.

   Here is an example of a JDBC trace file:

```
Session statistics: com.sap.db.jdbc.BasicChannelSession@6d3a388c
HanaConnectionFinalize@223aa2f7 10.7.167.208:30015
```

```
System:A00A00 TenantName:(none) SiteVolumeID:0:3
SiteType:NONE ConnectionID:347755 SessionID:1633354673470197
on sun.nio.ch.WindowsAsynchronousSocketChannelImpl[connected
local=/0:0:0:0:0:0:0:0:54958 remote=ykfl00620465a.amer.global.corp.sap/
10.7.167.208:30015]
  Duration:0 hours 0 minutes 0 seconds
  SentPacketCount:10 TotalTime: 7 ms
  ReceivedPacketCount:10 TotalTime: 736 ms
  SentByteCount:0.007 MB Uncompressed: 0.105 MB CompressionRatio: 14.045
  ReceivedByteCount:0.014 MB Uncompressed: 0.107 MB CompressionRatio: 7.637
```

Results

You have determined whether network compression is enabled for your connection and have discovered the compression rate of your connection.

Related Information

SQLDBC Tracing [page 785]
ODBC Tracing and Trace Options [page 656]

6.3.2 Reducing the Amount of Data Transferred on the Network

Reducing the amount of data that is transferred is especially beneficial if your network has low throughput (for example, many WANs).

These suggestions are unlikely to decrease performance on other networks. These tips include:

- Use stored procedures for large database queries to eliminate the need to send a large statement to the server across the network by allowing you to send a small CALL statement to execute the query.
- If you know that you will only be fetching the first row, or first few rows, of a query, then add a TOP clause to the query. The TOP clause prevents rows that you will not use from being transferred. Adding the TOP clause may also help the query optimizer know how to execute the query more efficiently.

Related Information

Network Compression [page 827]

6.3.3 Reducing the Number of Requests

Reducing the number of requests is particularly beneficial if your network has high latency (for example, most WANs).

The following recommendations may also improve performance on other networks:

- Where possible, move logic that requires many SQL statements from the application to one or more stored procedures or functions.
- Use batch inserts, updates, and deletes (also referred to as using arrays of parameter values) where possible. Batch operations change multiple rows in one request, instead of using one request per row.
- If your application retrieves many 1KB to 10KB LOB values, consider increasing the indexserver.ini > session > max_lob_prefetch_size server configuration parameter to around 5000 or 10000.
  - LOB values refer to column values or output parameters from a server LOB type, such as BLOB or NCLOB.

    > **Note**
    >
    > Depending on the application and LOB value sizes, increasing this value can drastically reduce the number of round trips and thus significantly improve performance. However, in some cases, the number of round trips and amount of data transferred can increase causing a decrease in performance. Whenever changing this value, performance testing of your applications is recommended.

  - This is a server-wide setting, so changes to this setting affect all applications and queries.
  - Increasing max_lob_prefetch_size far above 10000 is not recommended without significant performance testing.

6.4 Driver-Specific Tuning

6.4.1 Best Practices for Node.js

There are some best practices when using the SAP HANA Node.js driver (`@sap/hana-client`) which can help reduce poor application performance or poor scalability. Use the following suggestions to improve application performance:

- Due to the Node.js driver's `Statement.exec[ute]` or `Connection.exec[ute]` methods' return type, the entire result set must be fetched and processed even if the application only uses 1 or a handful of rows. For example, if XSA or a Node.js application executes a query that returns thousands or even millions of rows, the driver still needs to fetch and process the entire result set. If the application only accesses a small percentage of these rows this wastes time and results in poor performance. Using the TOP or WHERE clauses in queries to limit the number of rows returned is always recommended when only a small percentage of a result set is accessed. `Statement.exec[ute]` or `Connection.exec[ute]` refers to either the `exec` or `execute` method, both of which do the same thing.
- In cases where the entire result set (every single row returned by the query) is used by the application, the `Statement.exec[ute]` or `Connection.exec[ute]` methods have better performance than the `Statement.exec[ute]Query` method. However, in cases where a small number of rows (relative to

the size of the result set size) are fetched and processed, the `Statement.exec[ute]Query` method can have significantly better performance because only the rows fetched by the application need to be processed. If a SQL statement does not take parameters, `Connection.exec[ute]` is recommended over `Statement.exec[ute]` as it requires fewer round trips to the server.

- When using `Statement.exec[ute]` or `Connection.exec[ute]`, set the rowsAsArray option to **`true`** to improve the performance from the default. However, this also changes the format of the result set to an array of arrays. When set to **`false`**, the result set is an array of objects.
- If you are using `Statement.exec[ute]Query` and the ResultSet object, getting column values with one call to `ResultSet.getValues` performs better than multiple calls to `ResultSet.getValue`. Further, `getValues({asArray: true})` performs better than getValues with the default asArray false.
- Calling `ResultSet.getValues` synchronously instead of asynchronously can have significantly better performance as long as `getValues` does not block. When `ResultSet.getValuesCanBlock` returns false it is safe to use the synchronous `getValues` method with no risk of blocking the main Node.js thread.
- Preparing multiple statements (such as `Connect.prepare`) calls per second can create system congestion and slow performance. Instead, improve performance by preparing the statement once and executing it multiple times.
- Using `Statement.exec[ute]Batch` to process multiple rows (for example, multiple rows of INSERT or UPDATE statements) is more efficient than making multiple `Statement.exec[ute]` or `Connection.exec[ute]` calls.
- Always use a callback with Node.js driver methods that support callbacks. Failure to do so can result in the main Node.js thread being blocked while the client waits for a reply from the SAP HANA server, which significantly impacts scalability on concurrent requests.
- Avoid calling a Node.js driver method on the same connection that has an outstanding async callback method in progress. Failure to do so can result in the main Node.js thread being blocked while the client waits for a reply from the SAP HANA server. Methods that do not block the main thread are calling `Connection.abort()` and `Connection.state()` or calling a method with a callback.
- Increase the HDB_NODEJS_THREADPOOL_SIZE or the UV_THREADPOOL_SIZE environment variable. This value specifies the number of workers that can concurrently execute asynchronous method calls for different connections. Increasing this value may increase concurrency and reduce queuing of work.

Related Information

Node.js Application Programming [page 498]
exec[ute]([, Array][, Object][, Function]) Method [page 586]
exec[ute]Batch([Array][, Object][, Function]) Method [page 588]
getValuesCanBlock() Method [page 577]

6.4.1.1 Improve Node.js Performance with Connection Pooling

Connection pooling can improve Node.js performance if the driver is making multiple, brief connections to the database server.

If connection pooling is enabled for a connection, when it is disconnected, the connection is automatically cached and may be reused when the application reconnects. When an application opens a new pooled

connection, Node.js always searches the connection pool for an available connection with same connection string.

For connection pooling, use the `pooling` connection property. Node.js cleans up and reinitializes pooled connections.

To specify a maximum number of seconds that a connection can be cached in the pool, use the `connectionLifetime` connection property. A value of 0, which is the default, means that the connection is cached permanently.

If the application process connects again and there are cached connections available for the same connection string, then the cached connection is reused.

To ensure that connection pooling is transparent to the application, a connection is disconnected if a failure occurs when caching a connection.

If the lifetime of the connection has exceeded the specified connection lifetime, then the driver closes the connection.. The connection is reinitialized, and the cached connection continues to be connected to the database server even though the application has disconnected it. The cleanup and reinitialization of a connection includes rolling back all outstanding transactions, isolations levels, locations, and the schema. Temporary objects, such as temporary tables and procedures, are not reset when a cached connection is reused.

Related Information

Node.js Connection Properties [page 506]
Tuning an Application [page 820]

6.4.2 Best Practices for JDBC

Help reduce poor application performance or poor scalability using JDBC best practices.

Use the following JDBC connection properties to improve application performance:

closeHandlesOnFinalize/closeHandlesByCleaner

- Installs finalizers (JVM 8) or cleaners (JVM 9 and later) for each java.sql.Connection/Statement/ResultSet object, ensuring that the driver automatically sends a close request to the server if the application failed to do so before the object is garbage collected.
- The default is TRUE, which prevents JDBC applications from leaving connections, statements, and result sets open indefinitely on the server.
- Not required for programs that ensure java.sql.Connection/Statement/ResultSet.close() is invoked before the object is no longer reachable by any live threads.
- Setting to FALSE can significantly increase the performance of the JVM, especially under JVM 8, where finalizers are costly in terms of memory (much more so than cleaners under JVM 9 and later).

transactionalLobs

- Simulates LOB behavior as required by the JDBC specification by postponing the closing of statements and result sets that return CLOBs, NCLOBs, or BLOBs until the end of the current transaction.
- The default is TRUE, which allows an application to access a java.sql.Clob/NClob/Blob object from a closed prepared statement or closed result set within the current transaction.
- Required for some code generation frameworks which follow the specification strictly.
- Setting to FALSE can reduce the number of open statements and result sets on the server, reducing server memory requirements. Setting to FALSE can also decrease JVM memory requirements, reducing the frequency with which the garbage collector must run.

timeZonePerObject

- Instantiates a new java.util.TimeZone object for each prepared statement and result set for use in time-zone conversions.
- The default is TRUE, which allows the driver to react to time zone changes (such as when switching to and from daylight savings time) while the JVM is running.
- Rarely required (the ESRI test suite requires it).
- Setting to FALSE can significantly reduce the number of object instantiations in the JVM, reducing the frequency with which the garbage collector must run.

Related Information

JDBC Connection Properties [page 441]

6.5 Memory Tuning

Learn about packet caching and packet size and best practices for setting associated properties.

6.5.1 Turning off Packet Caching

By default, each connection created by SQLDBC-based HANA clients (ADO.NET, Go (golang), Node.js, ODBC, Python, Ruby) will keep a dedicated block in memory to be used for sending and receiving packets to avoid continually allocating and deallocating that memory with every request.

The size of this block will be equal to the packet size for that connection, which defaults to 1 Mb. For applications with a small number of connections, this will have minimal impact.

For applications with hundreds or thousands of connections, however, this can have a significant impact on memory consumption. To avoid this, it is recommended that applications needing large numbers of connections set the `packetCaching` connection property to `false`.

6.5.2 Increasing Packet Size

Increasing the packet size can reduce the number of round trips for these operations and can improve performance, but at the cost of increased memory usage on the client and the server.

If the application often processes numerous requests at the same time, so if your application is memory-sensitive, you may want to consider decreasing, or at least not increasing, the packet size (noting that there is a minimum size).

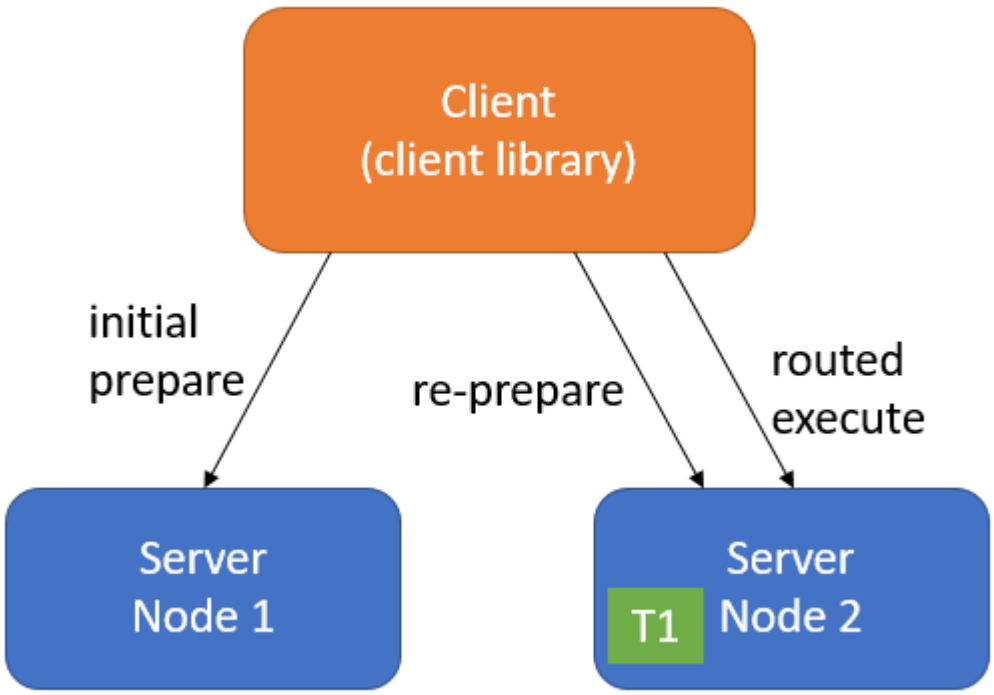
7 Statement Routing (SAP HANA Platform)

Statement routing is the method of evaluating the correct server node of a distributed system on which to execute a statement before statement execution. This reduces server processing overhead by minimizing communication between server nodes.

For SQLDBC-based client APIs and JDBC, the client library makes statement routing decisions to reduce server-side, inter-node routing because the client is the source of the request.

During preparation, table location information is collected and sent back to the client where it is cached. The client library then chooses an optimal host and then routes and executes the database request on that host. If the statement has not already been prepared on an optimal server node, then it must be prepared on a routed server node before it can be executed.

The following graphic shows how statement routing is implemented:



The following limitations exist for statement routing:

- Table locations are determined during preparation, so statement routing is only supported for prepared statements. If statements are not prepared, you cannot take advantage of statement routing.
  Use the routeDirectExecute connection property to force SQLDBC and JDBC to always use a separate prepare and execute, even if the application used a direct execute (an execute without a prepare).

  > ⓘ Note
  >
  > In some cases, applications or interfaces that use SQLDBC or JDBC (such as SAP HANA Cockpit, ABAP, or PyDBAPI) can perform a separate prepare and execute without the user's knowledge.

- The JDBC deferredPrepare connection property must be set to DISABLED (the default) to use statement routing.
- If accessed tables, partitions, or replicas are located on several different nodes then SAP HANA follows a round-robin strategy when selecting the node, based on a list of locations or partitioning information contained in the server, regardless of table size.
- If table replication causes statement routing issues, go to the SQL section of the `indexserver.ini` configuration file and set the use_new_placement_scheme_for_replicas property to FALSE.
- The server must send the client topology addresses for server nodes that are valid for the client's network.
- The use of topology must not be disabled due to the `ignoreTopology` connection property, port address mapping through a proxy, invalid format, or missing topology addresses.

Tracing for Statement Routing

Activate a database trace or user-specific statement routing trace on the debug level by executing the following:

```
indexserver.ini -> [TRACE] -> statementrouting = 'debug'
```

Preconditions

These preconditions indicate client-side feature settings.

JDBC and SQLDBC Statement Routing Settings

Server setting	Client Connection Property	Must Be Set To
indexserver.ini-> [distribution] -> client_distribution_mode	distribution	"Statement" (default) or "All"

SQLDBC Split Batch Client-Side Routing Settings

Server setting	Client Connection Property	Must Be Set To
indexserver.ini-> [distribution] -> split_batch_commands	splitBatchCommands	"1" (default)

Split Batch Insert

A split batch execution divides the array of parameters into groups of rows that are routed directly to the node where they are stored. For example, if a table is partitioned over three nodes and a batch execution is run with 300 rows of parameters that are distributed over all three nodes, then three separate routed executions are run, each with an average of 100 rows.

A split batch execution occurs when all of the following criteria are met:

- You are executing a prepared statement for an insert with parameters (for example: `INSERT INTO T1(C1,C2,C3) VALUES(?,?,?)`).
- The table being inserted into is partitioned with hash or range partitioning.
- Statement distribution is enabled.
- Split batch execution is enabled. This is enabled by default for SQLDBC-based clients through the splitBatchCommands connection property, and it is disabled by default for JDBC-based clients through the splitBatchCommands connection property.

Whole Batch Routing

When a partitioned table contains a unique constraint, batch splitting is automatically turned off. However, the application can still use statement routing by using the first row in the batch to determine the routing destination for the entire batch of parameters. This behavior can improve performance if the majority of the rows in the batch have the same optimal destination as the first row.

When the table is range partitioned, the application can break up a large batch of parameters into smaller batches where each batch belongs to the same node based on the parameter values and partition information. This improves performance behavior and is effective even when split batch execution is turned off.

The same optimization also exists for hash partitioning. The application has no easy way to know which partition a row should belong to because the internal hash function is not exposed to the application. The application can still make smaller batches that have the same hash partitioned parameter values. The performance gain in arranging parameters in smaller batches for range or hash partitioning is reduced as the batch size reduces.

Whole batch routing applies to SQLDBC-based SAP HANA drivers as well as the SAP HANA JDBC driver.

Related Information

JDBC Connection Properties [page 441]
ODBC Connection Properties [page 621]
Mapping Host Names for Database Client Access

7.1 Client-side Statement Routing for Hash Partitioning

If hash partitioning is used on a table, a client tries to connect directly with the server that holds the partitions matching the WHERE clause, if possible.

For example, if a table is partitioned with Hash partitioning on column A and the following SELECT statement is issued then the client will send the request to the node on which the partition is located for which matches to "A = 5" are expected.

```
SELECT * FROM mytable WHERE A = 5
```

This works only if the columns in the WHERE clause match the partitioning columns and are used in expressions with equality ("=").

This reduces the number of hops between the index servers and is especially important if you have a transactional (OLTP) workload. Therefore it should be a design goal to choose a partitioning scheme, that matches your queries. This also works for multi-level partitioning if Hash partitioning is used on the first level. This feature also works for DML.

> ⓘ Note
>
> For JDBC-based clients, optimal routing happens for prepared statements on Hash partitioned tables. To enable client-side statement routing for Hash partitioning, you must ensure that statement distribution is turned on.

7.2 Client-side Statement Routing for Range Partitioning (SAP HANA Platform)

Client-side routing for range-partitioned tables can improve operation performance by routing the request to the correct indexserver.

For example, if you create a column table with range-partitioning on column A and then issue the following SELECT statement, then the client sends the request to the node where the partition is located:

```
CREATE COLUMN TABLE PartitionTestTable (A TIMESTAMP) PARTITION BY RANGE(A)
   (PARTITION 20010101 <= VALUES < 20010102,
   PARTITION 20050101 <= VALUES < 20090101,
   PARTITION VALUES = 20140101,
   PARTITION OTHERS);
```

```
SELECT * FROM PartitionTestTable WHERE A = ?;
```

This optimization works only if the columns in the WHERE clause match the partitioning columns and are used in expressions with comparison predicates (=, >, <, >=, and so on). For more information about comparison predicates, see the *SAP HANA SQL and System Views Reference*.

Related Information

Comparison Predicates (ANY, SOME, and ALL)
Client-side Statement Routing for Hash Partitioning [page 839]

7.3 Configuring Statement Routing via SQL Hints (SAP HANA Platform)

Statement routing can be configured via SQL hints.

These hints can also be pinned to SQL statements.

DISTRIBUTION	Details
`ROUTE_BY('<table_name_1>', ..., '<table_name_n>')`	Routes the SQL statement to the hosts related to the specified table(s).
`ROUTE_BY_CARDINALITY('<table_name_1>', ..., '<table_name_n>')`	Routes the SQL statement to the host related to the specified table with the highest number of records.
`ROUTE_TO(<volume_id_1>, ..., <volume_id_n>)`	Routes the SQL statement to the hosts related to the specified volume IDs.
`NO_ROUTE_TO(<volume_id_1>, ..., <volume_id_n>)`	Excludes routing the SQL statement to the host related to the specific volume IDs.

7.4 Determining Statement Routing Details (SAP HANA Platform)

Statement routing details can be determined in a number of ways.

Access information about connection mappings to SAP HANA nodes:

- The M_CONNECTIONS system view

The following columns exist within SAP HANA system views and provide statement routing details:

System View	Column	Description
M_CONNECTIONS	CLIENT_DISTRIBUTION_MODE	Specifies the client distribution mode of the current connection. Can be specified as: OFF, STATEMENT_ROUTING, CONNECTION_DISTRIBUTION, or ALL.
	LOGICAL_CONNECTION_ID	Specifies the connection ID (the connection used to connect to the initial node before routing is performed).
M_CONNECTION_STATISTICS	EXECUTION_COUNT_BY_ROUTING	Specifies the number of executions based on client routing.
M_SQL_PLAN_CACHE	EXECUTION_COUNT_BY_ROUTING	Specifies the number of executions based on client routing.
	IS_DISTRIBUTED_EXECUTION	Specifies TRUE if tables are located on different nodes or services. Otherwise, specifies FALSE.
	TABLE_LOCATIONS	Specifies the locations of the involved tables.

Related Information

M_CONNECTIONS System View
M_CONNECTION_STATISTICS System View
M_SQL_PLAN_CACHE System View

7.5 Enabling SAP HANA Distribution Modes (SAP HANA Platform)

The distribution mode specifies how the initial client connection to the server is made.

Statement routing is the most commonly used SAP HANA client distribution mode.

Server Parameter

The global client distribution mode is defined with the following `INI` parameter:

```
indexserver.ini ( [distribution] client_distribution_mode = [ off | connection |
statement | all ] )
```

Possible values are:

Client distribution mode	Description
off	Specifies that there is no client distribution.
connection	Specifies that the initial connection node is chosen in a round-robin manner over all Coordinator and Worker nodes. For example, if the first application connection connects to Node1, then the second application connection connects to Node2, and so on. In Active/Active (read enabled) systems with takeover, the siteType connection property must be specified. Connection distribution is not available in SAP ABAP applications including SAP S/4 HANA and SAP BW/4HANA.
statement	Specifies statement routing based on table locations (the default).
all	Specifies concurrent activation of all distribution features.

When statement routing is enabled, check the distribution mode in the M_INIFILE_CONTENTS and the M_CONNECTIONS system views for each connection.

For optimal connection performance, the `distribution` connection property should be the same level as the server `client_distribution_mode`.

To enable the all distribution mode, set the distribution connection property to ALL.

Client Parameter

Statement routing and connection distribution can be configured based on the client distribution connection property:

```
Connection property ( distribution = [ OFF | CONNECTION | STATEMENT | ALL ] )
```

Distribution	Description
OFF	Specifies that there is no client distribution.

Distribution	Description
CONNECTION	Specifies that the initial connection that was made when the application established the connection is chosen in a round-robin manner over all Coordinator and Worker nodes. For example, if the first application connection has Node1 as its anchor, the second application connection would have Node2, and so on. In Active/Active (read enabled) systems with takeover, the siteType connection property must be specified. Connection distribution is not available in SAP ABAP applications including SAP S/4 HANA and SAP BW/4HANA.
STATEMENT	Specifies statement routing based on table locations (the default).
ALL	Specifies concurrent activation of all distribution features.

Specifying the `ignoreTopology`=TRUE connection property value also disables client distribution.

In SAP ABAP environments, implement this setting with the following SAP profile property:

```
dbs/hdb/connect_property = distribution = <distribution_mode>
```

Specify multiple connection properties in a comma-separated list. For example:

```
dbs/hdb/connect_property = distribution = STATEMENT, encrypt = TRUE
```

Configuration Between Client and Server

The distribution level used by a connection depends on both the client `distribution` connection property and server client_distribution_mode configuration:

Client	Server	Enabled Feature at Connection
*	off	Disabled
OFF	*	Disabled
CONNECTION	connection	Connection distribution enabled*
CONNECTION	statement	Disabled
CONNECTION	all	Connection distribution enabled*
STATEMENT	connection	Disabled
STATEMENT	statement	Statement routing enabled

Client	Server	Enabled Feature at Connection
STATEMENT	all	Statement routing enabled
ALL	connection	Connection distribution enabled*
ALL	statement	Statement routing enabled
ALL	all	Both enabled*

*Connection distribution is not available in SAP ABAP applications including SAP S/4 HANA and SAP BW/4HANA.

Related Information

Client Requirements For A Takeover (SAP HANA Platform)

7.6 Backoff Timer for Regular Statement Routing (SAP HANA Platform)

Versions 2.11 and newer of SQLDBC and JDBC drivers introduce a backoff timer for regular statement routing to minimize the effect of internal connection failures on routed statement executions.

With the backoff timer no new connections to a specific node for a statement routed execution are attempted until the timeout period elapses after a failed internal connection attempt to a specific node for a statement routed execution. This removes the routed statement execution performance penalty of trying to connect to the unavailable node by using the initial connection. Connections to the previously unusable internal node are available once the timeout period elapses.

The backoff timer starts with a minimum period of 60 seconds. The time period is doubled after each subsequent internal connection failure for the same node, up to a maximum of 3600 seconds. Every successful internal connection resets the time period to the minimum value.

> ⓘ Note
>
> Set the `statementRoutingFailureBackoff` connection property to FALSE to disable backoff timer.

Related Information

JDBC Connection Properties [page 441]
SQLDBC Connection Properties [page 795]

8 SAP HANA Differences from Other Vendors

SAP HANA has default behavior that is different from other vendors in some cases.

When you migrate an application and/or data from another database, you must account for these differences.

SAP HANA supports forward-only, insensitive, non-updatable cursors across all interfaces

8.1 Batch Inserts with Autocommit Enabled

A batch execute may be broken into one or more requests to the server.

Batch executes can be broken up due to the "split batch" distributed operation, or because the space for the parameters is more than the packet size. If auto commit is enabled for the connection, then each request (including each request in a single batch execute) is committed. If there is an error on one of these requests, then part of a batch execute may have already been committed before the error was reported.

If your application requires a batch execute to be atomic (all or nothing), then auto commit must be off before the batch execute and an explicit commit or rollback must be done after the batch execute.

8.2 Character Encoding

Internally, SAP HANA stores characters with a CESU-8 encoding and allows the storage of unpaired surrogates at the end of data values in columns.

The SAP HANA protocol also transmits characters using CESU-8 for all SQL statements.

Each of the client interfaces behave slightly differently, due to programming language differences:

- Java natively uses UTF-16 and converts to CESU-8
- SQLDBC converts to CESU-8 and supports multiple input encodings
- Ruby and Go are UTF-8 by default
- Node.js converts to UTF-8 from UTF-16
- ODBC uses UCS-2 or UTF-16 for Unicode function calls (with WCHAR parameters and host variable types)
- HDBSQL supports a UTF-8 encoding by default

8.3 String to VARBINARY, ARRAY, and BLOB Conversions

Different database vendors and different APIs have different behaviours for a client string to server VARBINARY/BLOB and server VARBINARY/BLOB to client string conversions.

Using a binary client data type instead of string is recommended when the server value is VARBINARY or BLOB to avoid potentially unexpected results.

For example, Node.js string values provided parameters are converted using HEXTOBIN when going to a VARBINARY server value (for instance, HEXTOBIN("12AB") results in the binary two bytes 0x12 0xAB), and to CESU-8 when going to a BLOB server value.

Other APIs may have different conversions.

ARRAY Data Type Support

HANA clients do not directly support the server ARRAY data type as a client type. When working with an ARRAY value, the data type is provided as a VARBINARY to the client.

8.4 Timestamps in SAP HANA

In some cases, SAP HANA handles timestamps differently than other databases. These differences must be accounted for in your application design.

SAP HANA only supports a TIMESTAMP column type that does not contain timezone information. Other database vendors have a TIMESTAMP without timezone column type and a TIMESTAMP with timezone column type.

String Literal Input Parameters

SAP HANA accepts string input parameters for TIMESTAMP columns that contain timezone information. The timezone information is dropped and the rest of the timestamp information is processed. This behavior may be different from other database implementations.

To preserve timezone data within TIMESTAMP columns in SAP HANA, design your application to either convert timestamps to UTC before adding the data or to assume that a timezone is being used for all TIMESTAMP columns by the application and clearly keep this assumption throughout the application.

Timestamp Object Input Parameters

Unless otherwise stated, each API that provides an interface to an SAP HANA server that allows you to insert a Timestamp object drops the extra timezone information provided via the object type without any adjustment to the date and time portions of data before insertion. These are the exceptions to this behavior:

Go (golang)

When a time.Time object is provided as an input parameter to a prepared upsert statement, then by default the SAP HANA golang driver (`SAP/go-hdb/driver`) converts the time.Time value to UTC before insertion. This behavior is different from other vendor's corresponding TIMESTAMP WITHOUT TIMEZONE column types: most other vendors drop the timezone information before insertion without adjusting the date and time information.

Use the SAP HANA golang driver's `utcTimeConversion` connection property to change the default behavior. The default behavior is TRUE.

```
db, err := sql.Open("hdb", "hdb://
<username>:<password>@<hostname>:<port>?utcTimeConversion=false")
```

JDBC

The setTimestamp() method has an override that allows for timezone adjustment via a calendar object. If this override is not used, then the driver sets the timezone using the calendar of the virtual machine that is running the application.

Node.js

The SAP HANA Node.js driver returns an error when a Date object is supplied. To work around this behavior, convert the Date object to a string (by calling toISOString() on the object) and supply the string as the parameter.

Ruby

The SAP HANA Ruby driver returns an error when a Time object is supplied as an input parameter to a prepared statement. To work around this behavior, call to_s or strftime to convert the Time object to a string and use the created string as the input parameter.

SAP HANA Data Provider for Microsoft ADO.NET

The SAP HANA Data Provider returns an error when a DateTimeOffset input parameter is used to insert data into a TIMESTAMP column. To work around this behavior, convert the DateTimeOffset object to a DateTime object (by accessing the UtcDateTime property on the original object) and add the DateTime object as the parameter.

9 SAP HANA User Store (hdbuserstore)

The SAP HANA user store (hdbuserstore) is a tool installed with the SAP HANA client. Use it to store connection information to SAP HANA systems on the client so that client applications can connect to SAP HANA without users having to enter this information. It is typically used by scripts connecting to SAP HANA.

The SAP HANA user store allows you to store SAP HANA connection information, including user passwords, on clients. In this way, client applications can connect to SAP HANA without the user having to enter host name or logon credentials. You can also use the user store to configure failover support for application servers in a 3-tier scenario (for example, SAP Business Warehouse) by storing a list of all the hosts that the application server can connect to.

The SAP HANA user store is installed with the SAP HANA client package. After you install the SAP HANA client, the `hdbuserstore` program is located in one of the following directories:

- `/usr/sap/hdbclient` (Linux/UNIX)
- `%SystemDrive%\Program Files\sap\hdbclient` (Microsoft Windows)

The store runs on all platforms supported by SAP HANA client interfaces and SAP BASIS 7.20 EXT.

Managing the Secure Store

Connection information stored in the SAP HANA user store is saved in the secure store file `SSFS_HDB.DAT`.

Operating system	Default location
Microsoft Windows	`<%ALLUSERSPROFILE%>\.hdb\<host-identity>\<security-identifier(SID)>\SSFS_HDB.DAT`
UNIX	`<home-directory-of-effective-user-ID>/.hdb/<host-identity>/SSFS_HDB.DAT`

You can control the absolute or relative store location for both platforms by using the `HDB_USE_STORE_PATH` environment variable. For more information, see Environment Variables [page 20].

For Microsoft Windows systems:

- `<%ALLUSERSPROFILE%>` stores a path to profile settings for all users on the system. On Microsoft Windows Vista and later, the default value for this environment variable is `C:\ProgramData`.
- `<host-identity>` is determined as follows:
  - If `%HDB_USE_IDENT%` is set, its value determines `<host-identity>`. `%HDB_USE_IDENT%` must only contain alphanumeric characters, underscores (_), or dashes (-). If any other character is included in the value, the variable is ignored.
  - If `<computer-name>` (as returned by `GetComputerName()`) is set, its value determines `<host-identity>`.

- `<security-identifier(SID)>` is the Windows Security Identifier of the user that created the Secure User Store.
- When connecting using a client, if the `$SAPLOCALHOST` environment variable is set, then its value determines `<host-identity>`.

For UNIX systems:

- `<host-identity>` is determined as follows:
  - If `$HDB_USE_IDENT` is set, its value determines `<host-identity>`. `$HDB_USE_IDENT` must only contain alphanumeric characters, underscores (_), or dashes (-). If any other character is included in the value, the variable is ignored.
  - When storing or listing keys using the `hdbuserstore` tool, if `<virtual-hostname>` is set by using the -H command line option, then its value determines `<host-identity>`.
  - The computer's `<hostname>` determines `<host-identity>`.
  - When connecting using a client, if the `$SAPLOCALHOST` environment variable is set, then its value determines `<host-identity>`.
  - On Linux only, if `<virtual-hostname>` is specified on the client install command line (by using the -H flag with hdbinst), its value determines `<host-identity>`.

If the `SSFS_HDB.DAT` path does not already exist, then the `hdbuserstore` program creates it upon store write operation. Read operation would result in an error if the path doesn't exist.

The secure store's content is platform-dependent. You cannot copy the secure store from one platform to another.

Managing Connection Information

Use the `hdbuserstore` program to store and manage connection information in the secure store. For more information about the available commands, see *hdbuserstore Commands*.

The secure user store is user specific, so only the operating system user who owns the corresponding secure store file can access the secure store. However, it is possible, with the appropriate operating system privileges, to manage another user's secure store. This behavior is needed, for example, to manage the connection details for ABAP on Microsoft Windows since the application server is running under a different user (SAPService`<SAPSID>` instead of `<SAPSID>`adm).

Using Stored Connection Information

When the secure store is accessed in the context of the correct operating system user, you can open it with a user key.

Client	How to Connect Using a Stored User Key
HDBSQL	Specify the key to be used with the -U or -Z connection option: `hdbsql -U <KEY>` or `hdbsql -Z KEY=<KEY>`

Client	How to Connect Using a Stored User Key
ODBC	Specify the user store key with the @ sign in your data source:`servernode=@<KEY>`
ABAP	Specifies the key DEFAULT by default
JDBC	Specify the `<KEY>` and `<VIRTUALHOSTNAME>` properties to read the secure store directly
Node.js	Specify the user store key with the @ sign in your data source:`servernode=@<KEY>`
Python	Specify the `<USERKEY>` (case sensitive) or the `<KEY>` (case insensitive) in the dbapi.connect function. For example, `dbapi.connect(<USERKEY>=<KEY>)`
Go	Specify the `<KEY>`. For example, `hdb://user:password@localhost:30015?<KEY>=<VALIDKEY>`
ADO.NET	Specify KEY=`<key-name>` to access the secure store.

To access the store using JDBC, there are two connect options: `key` and `virtualHostName`. `key` is the `hdbuserstore` key that you use to connect to SAP HANA, while `virtualHostName` specifies the virtual host name. This option allows you to change where the `hdbuserstore` searches for the data and key files.

To connect, define the `hdbuserstore` key using the `key` connect option. JDBC only supports reading the key and data files for existing keys and using those keys to connect to SAP HANA.

Managing the Encryption Key

The initial default encryption key of the secure user store is automatically changed when the first entry is created.

This key is automatically changed the first time the `SET` or `DELETE` commands are executed. You can also change it explicitly by using the `CHANGEKEY` command. The SET and DELETE commands implicitly execute the CHANGEKEY command. For more information, see *Change the Secure User Store Encryption Key* in the *SAP HANA Administration Guide*.

If a user forgets the stored password, then you cannot recover that password because the system does not display passwords in a human-readable form.

Related Information

SAP HANA Client Installation and Update Guide
Environment Variables [page 20]

9.1 hdbuserstore Commands

Several commands are available for managing connection information stored in the secure user store of the SAP HANA client (`hdbuserstore`).

You store and manage connection information in the user store with the `hdbuserstore` program. Execute commands using the following syntax:

```
hdbuserstore [OPTION]... COMMAND [PARAMETER]...}}
```

Returned Codes

Every command returns 0 if successful.

If an error occurs, every command returns a positive value, which:

- Has a value of 1 under exceptional circumstances (usually, this is the error).
- Has a value of 100 (SQLDBC_NO_DATA_FOUND) if you use a command that references a KEY that is supposed to be found in the store, but the tool cannot find it. This applies to the `LIST` and `DELETE` commands.

Note that a return value of 1 is possible for exceptional circumstances, including invalid command input, store corruption, and read/write errors. Use `-v LIST` without <KEY> arguments to check the health of the store when exceptional circumstances are not related to invalid command input.

Command Options

Option	Description
-h	Displays a help message.
-H <HOST>	Assumes the host name <HOST>. **Note** This option is implemented for non-Microsoft Windows only. Alternatively, you can set the HDB_USE_IDENT environment variable.
-i	Enables interactive mode.

Option	Description
-u <USER>	Perform operations for other operating system users. **Note** You must have administrator privileges to work on the store of a different user. This option is for Microsoft Windows administrators only.
-v	Execute commands in verbose mode.

Commands

Command	Parameter	Description
HELP	-	Displays a help message.
LIST	KEY	Lists entries with the specified key, even without any arguments. Passwords are not displayed. Returns a value of 100 (SQLDBC_NO_DATA_FOUND) if the tool cannot find the KEY in the store.
DELETE	KEY	Deletes entries with the specified key. Returns a value of 100 (SQLDBC_NO_DATA_FOUND) if the tool cannot find the KEY in the store.
SET	KEY	Sets the entry key. **Note** Underscores, hyphens, periods, and alphanumerical characters (0-9, a-z, A-Z) are allowed. The resulting key is always upper case (0-9, A-Z).
	ENV	Sets the connection environment in the format: • <host:port>
	USERNAME	Sets the user name for the profile.

Command	Parameter	Description
	PASSWORD	Sets the password for the profile. **Note** We recommend executing the SET command in interactive mode so that you are prompted to enter the password. If you enter the password directly in the command, it is stored in your shell's command history.
SetX509	-	Stores the connection property authenticationX509. The SQLDBC-based driver using such user key performs X.509 authentication. **Note** When listing the key, the hdbuserstore does not connect to the SAP HANA to resolve the database user name bond with the certificate. It prints `X.509 USER` whenever the key is set with the `SetX509` command.
	KEY	Sets the entry key. **Note** Underscores, hyphens, periods, and alphanumerical characters (0-9, a-z, A-Z) are allowed. The resulting key is always upper case (0-9, A-Z).
	ENV	Sets the connection environment in the format: • `<host:port>`
	CERT_FILE	Sets the client certificate file path, with a maximum length of 128 characters.
	DATABASE	Sets the database name in MDC environment.
CHANGEKEY	-	Randomly generates a new master encryption key and re-encrypts password of all keys with the new master key. This command compacts the store by removing records that are marked as deleted.

Command	Parameter	Description
COMPACT	-	Permanently delete entries already marked as deleted.
ADDFROMDIR	DIR	Adds entries from a store specified by the DIR parameter to the secure user store without overwriting existing keys. You must have read permission for both the SSFS_HDB.DAT and SSFS_HDB.KEY files in the specified directory. **Note** This command does not copy trace configurations stored in the SSFS_HDB.DAT file. If this command reads a key name in the store and the same key name is already in use, then the existing key remains unchanged. If this command is interupted, then the keys that have already been added are not automatically rolled back. An intermittent store backup file is created for manual recovery. The backup file is created before each key is added and is automatically removed after each successful add single key operation. Returns SQLDBC_NO_DATA_FOUND if the store located in the directory DIR has no keys in it.
LISTFROMDIR	DIR	Lists entries from a store in the store directory. You must have read permission for both the SSFS_HDB.DAT and SSFS_HDB.KEY files. Returns SQLDBC_NO_DATA_FOUND if the store located in the directory DIR has no keys in it.

Examples

Action	Command	Example
Create a user key in the user store	`hdbuserstore SET <KEY> <host:port> <USERNAME> <PASSWORD>`	`hdbuserstore SET millerj "localhost:30115" JohnMiller 2wsx\$RFV`
List all available user keys (passwords are not displayed)	`hdbuserstore LIST <KEY>`	`hdbuserstore LIST millerj` The following information is displayed: • `KEY: millerj` • `ENV: localhost:30115` • `USER: JohnMiller` or (if `SetX509` was specified): • `KEY: millerj` • `ENV: localhost:30115` • `X.509 USER`
Configure failover support for an application server by specifying a list of a list of host-port pairs that the server can connect to	`hdbuserstore SET DEFAULT "<hostname_node1>:3<inst>15; … ;<hostname_node(n)>:3<inst>15" <sapsid> <password>`	`hdbuserstore SET default "ld9490:33315;ld9491:33315;ld9492:33315;ld9493:33315" SAPP20 <Password>`

Related Information

Environment Variables [page 20]

9.2 Change the User Store Encryption Key

If you are using the current version of the SAP HANA client, there is no need to change the encryption key of the secure user store. However, if you are using an older version of the SAP HANA client, we recommend changing the encryption key after installation of the SAP HANA client.

Procedure

1. Change the encryption key with the command:

```
hdbuserstore CHANGEKEY
```

The `hdbuserstore` program is available after installation of the SAP HANA client in the following directories:

- `/usr/sap/hdbclient` (Linux/Unix)
- `%SystemDrive%\Program Files\sap` (Microsoft Windows)

A new master encryption key is randomly generated and data in the secure store is re-encrypted with the new key.

2. Verify that the key has been changed with the command:

```
hdbuserstore LIST
```

If the key file `SSFS_HDB.KEY` exists, the time stamp of the file indicates when the key was last successfully changed.

Related Information

Encryption Keys and Files
Client-Side Data Encryption

9.3 Use the User Store (hdbuserstore)

The SAP HANA user store enables you to store information that is used to connect to an SAP HANA system.

Context

To avoid entering connection-related information manually each time you want to establish a connection to an SAP HANA database, store the connection information in the user store. As part of the configuration, you assign a key and use this key when making connections. The user store is located on the system running the client interface.

Connection information stored in the secure store is saved in the (per user) secure store file `SSFS_HDB.DAT`. For example, the default location on Linux is: `$HOME/.hdb/<hostname>/SSFS_HDB.DAT`.

Storing the connection information makes it easier to move between systems (for example, when executing SQL from the command line), and also keeps connection information, including user names and passwords, in a secure place.

The SAP HANA user store is part of the client installation and is compatible with all SQLDBC- and JDBC-based connections included in the installation.

Procedure

1. Run the following command:

   ```
   hdbuserstore.exe set <key> <server>:<port>@<database> <user> <password>
   ```

   The server, port, database name, user name, and password are now stored in the user store. `@<database>` is optional if `<server>:<port>` specifies the database location directly. The key is a string that you use to refer to this set of connection information.
2. Use the key to reference a connection to a particular system. Always precede the key with an `@` (for example, `@mykey`).

   Use the example key in the following ways:

   - In the connection string, use the key for the `serverNode` parameter and do not include a user name and password. For example:

     ```
     serverNode={@mykey};driver={hdbodbc};
     ```
   - To create an ODBC data source, enter the key (for example, `@mykey`) for the server and port.
   - To test your connection on Microsoft Windows (by running `odbcreg -t HDBODBC`), use the example key `@mykey` for the server and port. If the connection is successful, then your results look similar to the following:

     ```
     odbcreg -t HDBODBC
     MDAC version: 6.3.9600.16384
     ODBC driver test for 'HDBODBC'.
     retcode: 0
     outString(35): serverNode={@mykey};driver={HDBODBC};
     ```

     The response includes a sample connection string using the key.

Related Information

Secure User Store (hdbuserstore)
Test the ODBC Installation on Microsoft Windows

10 SAP HANA HDBSQL (Command-Line Reference)

SAP HANA HDBSQL is a command-line tool for executing commands on SAP HANA databases.

Related Information

SAP HANA Platform: Getting Started With the SAP HANA Database Explorer
SAP HANA Cloud: Getting Started With the SAP HANA Database Explorer

10.1 SAP HANA HDBSQL Options

Set information for the database and for database objects by using SAP HANA HDBSQL commands.

In addition to SAP HANA HDBSQL commands, you can also enter an SQL statement or a database procedure. The statement or procedure must be enclosed in quotation marks.

Configuration Options

Use the following options to modify the operation of SAP HANA HDBSQL commands.

Database Session

Option	Description
-d `<database-name>` (SAP HANA Platform only)	Specifies the name of the multitenant database container in a multiple-container system.
-e	Specifies that encrypted data transmission is used.
-n `<host>`[:`<port>`]	Specifies the name of the computer on which the system is installed and, optionally, the port number. The default is *localhost:30015*.
-p `<database-user-password>`	Specifies the password for logging on to the database.
-proxyhost `<hostname>`	Connects to a SOCKS5 proxy located at the `<hostname>`.

Option	Description
-proxyhttp (SAP HANA Cloud only)	When set to TRUE, enables HTTP proxy authentication and implies that ENCRYPT=TRUE. This property is only applicable if proxyHostname is specified and webSocketURL is not specified.
-proxypassword `<pwd>`	Authenticates against a SOCKS5 proxy with `<pwd>`.
-proxyport `<port>`	Connects to a SOCKS5 proxy using `<port>`. The default is 1080.
-proxyuserid `<userid>`	Authenticates against a SOCKS5 proxy using `<userid>`.
-r	Directly executes SQL statements instead of using the default prepared statements.
-S `<sql-mode>`	Specifies the SQL mode: INTERNAL/SAPR3.
-saml-assertion `<file>`	Uses a file to provide a SAML assertion.

-stdin	Reads SAP HANA HDBSQL options from the standard input, for example:
	`hdbsql -n host -u user -stdin < secrets.txt`
	In this example, secrets.txt contains sensistive information, for example:
	`-p secretpassword -sslprovider openssl`

-u `<database-user>`	Specifies the user name for logging on to the database.
-U `<user-store-key>`	Uses credentials from the user store.

Option	Description
-V `<variable-definition>`[,...] [*prompt*] | [*noprompt*]	Specifies a variable. The `<variable-definition>` can either be: **Explicit** Explicitly defines the variable using `<variable-name>=<variable-value>`, with no spaces between the variable name, the equal sign (=), or the value. **Positional** Defines the variable according to its position on the command line by using `<variable-value>`. The variable declaration's position on the command line is relative to other positional variables and determines which parameter it is replacing. For example, `<value 1>`, `<value 2>` means that &1 in the SQL script is replaced by `<value 1>`, and &2 is replaced by `<value 2>`. Spaces are only allowed if the entire list of definitions are quoted and any special characters are escaped. Specify *[prompt]* to be prompted for a definition whenever an undefined variable is encountered. Specifying *[noprompt]* means that undefined variables are ignored and the command may fail if it contains undefined variables. The default is *[noprompt]*.
-wsurl (SAP HANA Service only)	Specifies the WebSocket URL used to connect to a single-node SAP HANA server. WebSockets connections have the following default connection property settings: • encrypt=TRUE • compress=TRUE • ignoreTopology=TRUE • reconnect=FALSE
-z	Switches off AUTOCOMMIT mode.
-Z`<value>=<property>`	Sets SQLDBC connection options. For example, `SQLMODE=SAPR3`.

Input and Output

Option	Description
-br `<reset-command>`	Specifies a batch reset command (for example, RESET), which tells SAP HANA HDBSQL to ignore the most recent query that is being constructed to send to the server. The *-br* command should not be an SQL statement or the command separator. When a *-br* command is specified, it and all command separators must also be placed by themselves on the line following the command. Specifying the *-br* command on the same line as the command separator results in an error since it is not a valid query.
-c `<separator>`	Specifies the separator used to separate individual commands when importing commands from a file. The default value is `;`.
-history <#>	Specifies the number of items to keep in the history buffer. The default is 50.
-I `<file>`	Imports commands from a batch file. Separate individual commands using a separator. The default value is `;`.
-m	Activates multiple-line mode for entering SAP HANA HDBSQL commands.
-nochop	Specifies not to remove trailing blanks from a character output.
-o `<file>`	Writes the results to a file.
-O `<file>`	Appends the results to an existing file. If there is no file, it writes the results to a new file.

Option	Description
-optionsfile `<file>`	Specifies that command-line options must be listed in a one option per line format in the file. The following rules apply: • Blank lines in the options files are allowed. • Blank options are allowed. • Dashes are required in front of all options files. • Any options arguments are allowed and must follow the dashed option on the same line. • Spaces between the option and any arguments are optional. • SAP HANA HDBSQL displays the applied option in a standard output. • SAP HANA HDBSQL displays the the applied arguments, if any, in a standard output only if they are in debug mode (for example, *-t*). • An optional Byte Order Marker at the beginning of the option file is supported. It can be used to select UCS2 Big Endian, UCS2 Little Endian, and UTF8 encoding. The default is UTF8 if no BOM is specified. • Multiple *-optionsfile*s in the command line are allowed but repeating the same file is not allowed. • The *-optionsfile* options inside the file support chaining, however circular chaining is not allowed.
-printoutput { *OFF* | *MESSAGE* }	Specifies how to handle the SQLSCRIPT_PRINT library output. The parameters are case sensitive. MESSAGE — This is the default setting if you do not specify *-printoutput*. MESSAGE causes the SQLSCRIPT_PRINT library output to be printed on standard output. The *-x* option suppresses the SQLSCRIPT_PRINT library output even if **`-printoutput MESSAGE`** is set. OFF — OFF suppresses output from the SQLSCRIPT_PRINT library when you call a stored procedure.
-qto | *-querytimeout*	Sets a server-side timeout for all SQL operations, in seconds. If any SAP HANA HDBSQL SQL operation exceeds the maximum timeout value on the server, then it is canceled with a server error message. Setting a timeout value of 0 disables the timeout (default).
-quiet	Hides the SAP HANA HDBSQL welcome banner.
-resultencoding `<encoding>`	Forces output encoding for result data: **`UTF8`**, **`LATIN1`**, or **`AUTO`** (the default).

Option	Description
-separatorownline	Deprecated, do not use.
-strictSeparatorLine	Removes the parsing of a single quote, double quote, and BEGIN...END nesting. Separator line matching is strict and no leading or trailing spaces are allowed. When *-c* is not used, the default separator is a semi-colon on its own line. Batch tracing (*-f* or *-fn*) also produces output for comment-only batches even when they are not sent to the server.
-x	Suppresses additional output, such as the number of selected rows in a result set.

Formatting Output

Option	Description
-A	Returns the result set in an aligned format.
-a	Suppresses the output of the column names in the result set.
-b `<maximum-length>` | *all*	Defines the maximum number of characters for the output of LOB values. The default value is 32 bytes. Specifying *-b* | *all* displays the whole binary length. The argument value is case insensitive.
-C	Suppresses the escape output format.
-f	Returns all SQL statements that are sent to the database instance.
-fn	Returns all SQL statements that are sent to the database instance and formats them with numbered lines. Numbered lines make it easier to determine on which file line a potential error has occurred.
-F `<separator>`	Specifies which string SAP HANA HDBSQL uses as a separator between the individual columns of the result set. The default value is |.
-g `<null-value>`	Specifies the character for NULL values in the result set. The default value is ?.
-j	Switches off the page-by-page scroll output. In the interactive prompt, the result output is always paged to the utility specified by the PAGER environment variable, or the "more" utility ("less" on macOS) if unspecified.
-nocmdstats	Suppresses printing of the number of rows affected and time taken by each SQL command executed.

Option	Description
-oldexectimes	Uses SAP HANA 1.0 execution-only timing. SAP HANA HDBSQL in SAP HANA 1.x only reports time for client and server executions, not fetches for result sets. As of SAP HANA 2.0, SAP HANA HDBSQL includes times for executions and fetches by default.
-p `<prefix>`	Use `<prefix>` as the row prefix for printing. The default value is `|`.
-q `<suffix>`	Use `<suffix>` as the row suffix for printing. The default value is `|`.
-Q	Outputs each column of the result set in a new row.
-timePerBatch	Prints the overall and server time taken by each SQL command executed in batch mode using option *-I*. Option *-x* does not suppress the output of *-timePerBatch*.
-totalBatchTime	Prints the overall and server time taken in total by all SQL commands executed in batch mode using option *-I*. Option *-x* does not suppress the output of *-totalBatchTime*.

> **Note**
>
> By default, SAP HANA HDBSQL removes whitespace of the end of column data values. To prevent this, add `-Z CHOPBLANKS=0` to the SAP HANA HDBSQL command-line.

Other

Option	Description
-E	Specifies the `<exit-code>` if HDBSQL returns an error. Values are: OKAY = 0, ERROR = 1, USAGE = 2, SQLERROR = 3, and VAR = 4.
-h	Displays the most commonly used help options.
-h2	Displays all help options.
-t	Outputs debug information.
-T `<file>`	Activates the SQLDBC trace, which writes the trace data to the specified file.
-v	Displays version information about the SAP HANA HDBSQL program.

SSL Options

Option	Description
-sslcreatecert	Creates a self-signed certificate.

Option	Description
-sslhostnameincert `<hostname>`	Specifies the hostname of the server for which the certificate has been granted. The default is the *FQDN* if the *-n* option has not been specified.
-sslkeystore `<key-store-file>`	Specifies the SSL keystore name.
-sslprovider `<provider>`	Specifies the cryptographic service provider that is used for SSL connection: commoncrypto, openssl, or mscrypto.
-sslsniname `<sni-name>`	Specifies the name of the host that is attempting to connect at the start of the TLS handshaking process.
-ssltrustcert	Skips certificate validation.
-ssltruststore `<trust-store-file>`	Specifies the SSL truststore name.

Interactive Options

Use the following options when operating SAP HANA HDBSQL in interactive mode.

Command	Description
\? *\h[elp]*	Displays all SAP HANA HDBSQL commands.
\a[utocommit] [*ON*|*OFF*]	Switches AUTOCOMMIT mode on or off.
\al[ign] [*ON*|*OFF*]	Controls whether SQL statement results are formatted.
\c[onnect]	Logs a user onto the database.
\cmdstats [*ON*|*OFF*]	Switches printing of the number of rows affected and time taken by each executed SQL command on or off. This command does not affect the output of batch option *-I*. Option *-x* overrides *\cmdstats*.

Command	Description
\dc [<pattern>]	Lists all table columns that correspond to the specified [<pattern>] and to which the current user has access. [<pattern>] is specified as follows: [<schema>.] [<object-name>]. The following placeholders are supported: • For one character: _ • For any number of characters: % If a pattern is not specified, then the system returns information about all table columns to which the current user has access. This command returns the following information: • Column name • Data type • Column length • Null value permitted or not • Position of the column in primary key of table (if applicable)
\de [<pattern>]	Lists all the indexes of database objects that correspond to the specified [<pattern>]. [<pattern>] is specified as follows: [<schema>.] [<object-name>]. The following placeholders are supported: • For one character: _ • For any number of characters: % If a pattern is not specified, then the system returns information about all indexes for database objects to which the current user has access. This command returns the following information: • Index name • Columns contained in the index • Position of column in the index • Specifies whether the index is UNIQUE • Sort sequence
\di[*sconnect*]	Logs the user off of the database.

Command	Description
\dp [<pattern>]	Lists all database procedures that correspond to the specified [<pattern>]. [<pattern>] is specified as follows: [<schema>.] [<object-name>]. The following placeholders are supported: • For one character: _ • For any number of characters: % If a pattern is not specified, then the system returns information about all database procedures to which the current user has access. This command returns the following information: • Schema name • Name of the database procedure • Package to which the database procedure is assigned
\ds [<name>]	Lists all schemas that correspond to the specified [<name>]. [<name>] is specified as follows: [<schema>.] [<object-name>]. The following placeholders are supported: • For one character: _ • For any number of characters: % If a pattern is not specified, then the system returns information about all schemas to which the current user has access. This command returns the following information: • Schema Name • Owner
\dt [<pattern>]	Lists all tables that correspond to the specified [<pattern>]. [<pattern>] is specified as follows: [<schema>.] [<object-name>]. The following placeholders are supported: • For one character: _ • For any number of characters: % If a pattern is not specified, then the system returns information about all tables to which the current user has access. This command returns the following information: • Schema name • Table name • Table type

Command	Description
\du [<name>]	Lists all database users that correspond to the specified [<name>]. [<name>] is specified as follows: [<schema>.] [<object-name>]. The following placeholders are supported: • For one character: _ • For any number of characters: % If a name is not specified, then the system returns information about all database users to which the current user has access. This command returns the following information: • Name of the database user • User properties
\dv [<pattern>]	Lists all views that correspond to the specified [<pattern>]. [<pattern>] is specified as follows: [<schema>.] [<object-name>]. The following placeholders are supported: • For one character: _ • For any number of characters: % If a pattern is not specified, then the system returns information about all views to which the current user has access. This command returns the following information: • Schema name • View name • View types
\e[dit] [<file>]	Writes the command buffer to the specified file where you can edit it with an editor.
\es[cape] [ON|OFF]	Switches the escape output format on or off.
\f[ieldsep] <separator>	Uses the specified separator character to separate the individual fields of the result. The default is `,`.
\g	Executes the commands in the command buffer and returns the results.
\hi[story] <#>	Specifies the number of items to keep in the history buffer. The default is 50.
\i[nput] <file>	Imports commands from the specified batch file. Separate individual commands using a separator. The default value is `;`.
\m[ode] { INTERNAL | SAPR3 }	Changes the SQL mode.
\mu[ltiline] { ON | OFF }	Switches multiple line mode on or off.
\o[utput] <file>	Redirects the result to a file.

Command	Description
\p[rint]	Displays the current command buffer.
\pa[ger]	Displays results consecutively, not page by page.
\q[uit]	Exits SAP HANA HDBSQL.
\qto | \querytimeout	Sets a server-side timeout for all SQL operations, in seconds. If any SAP HANA HDBSQL SQL operation exceeds the maximum timeout value on the server, then it is canceled with a server error message. Setting a timeout value of 0 disables the timeout. This is the default.
\r[eset]	Deletes the current command buffer.
\read <file>	Reads commands from the specified batch file. Separate individual commands using a separator. The default value is ;.
\ro[wsep] <separator>	Uses the specified separator character to separate the individual rows of the result.
\s[tatus]	Displays general information about the database.
\serverstats | \sstats	Retrieves server statistics, including CPU time, processing time, and memory usage.
\vc	Clears all of the currently defined variables.
\vd <variable-name> <value>	Defines a <variable-name> to be replaced by a <value>, or replaces the value stored with a new value if the <variable-name> is already is defined.
\ve { ON | OFF }	Switches the variable escaping on or off.
\vl	Lists all the currently defined variables.
\vp { ON | OFF }	Switches variable prompting on or off.
\vs { ON | OFF }	Switches variable substitution on or off.
\vu <variable-name>	Removes the definition from the specified variable.

Related Information

SQLDBC Connection Properties [page 795]

10.2 Log On to a Database

Log on to the database as a database user to use SAP HANA HDBSQL interactively and to execute commands.

Prerequisites

The user logging on must be a database user. If you do not specify the user name and password of a database user, then the logon is attempted using Kerberos authentication.

Procedure

- Log onto a database using SAP HANA HDBSQL with either a one-step or two-step process.
  a. To log onto a database in one step, with a user name and password, run one of the following commands:

Option	Action
Log onto a database in a single-container system	Run the following command all on one line: `hdbsql -n <host> -i <instance> -u <database_user> -p <database_user_password>`
Log onto a database in a multitenant database container	Run the following command all on one line: `hdbsql -n <host> -i <instance> -u <database_user> -p <database_user_password> -d <database_name>`

  b. To log onto a database in two steps, with a user name and password, run the following commands:
     1. Start SAP HANA HDBSQL by running `hdbsql`.
     2. Log on to the database by running the following:

Option	Action
Log onto a database in a multitenant database container	Run the following command all on one line: `\c -n <host> -i <instance> -u <database_user> -p <database_user_password> -d <database_name>`

> **Note**
>
> You can log on with user credentials for the secure user store (`hdbuserstore`) with -U `<user_key>`. For more information, see *Secure User Store (hdbuserstore)* in the *SAP HANA Security Guide*.

Results

The user is connected to the database.

10.3 Run Commands

Run SAP HANA HDBSQL commands in interactive and non-interactive mode.

Context

To execute an SQL statement or a database procedure as a command, place the statement or procedure in quotation marks.

> **Note**
>
> SAP HANA HDBSQL supports input parameters in interactive mode only and will prompt for input for the parameter. The following statement will not work outside of interactive mode:
>
> ```
> SELECT * FROM tab WHERE col3 = ?
> ```
>
> For similar use cases, use substitution variables. See Substitution Variables [page 876] for more information.

Procedure

- Run a command in interactive (session) mode as follows:
  a. Call SAP HANA HDBSQL by running the following command: `hdbsql`
  b. Type in the command and press **`Enter`**. If prepared statements are toggled on and the command contains input parameters, SAP HANA HDBSQL will prompt for each parameter. Type in the parameter data at each prompt and press **`Enter`**
  SAP HANA HDBSQL runs the command.
  c. Exit SAP HANA HDBSQL by running one of the following commands: `exit | quit | \q`

- Run a command in non-interactive (command) mode as follows:

  ```
  hdbsql [options] <command>
  ```

  SAP HANA HDBSQL runs the command and then exits.
- Run multiple commands from a batch file as follows:

  ```
  hdbsql [<options] -I <file>
  ```

  SAP HANA HDBSQL imports the commands from the specified file and processes them in the background. Specify the separator used in the batch file to separate individual commands by using the -c `<separator>` command line option. The default value is a semicolon (;).

  > **Note**
  >
  > If you run commands from a batch file, then AUTOCOMMIT mode is activated by default. If you deactivate AUTOCOMMIT mode, then the batch file must contain an explicit COMMIT statement to ensure that SAP HANA HDBSQL executes the SQL statements immediately after the batch file has been imported.

Example

Run the following command to display general information about the database in command mode with simultaneous database logon:

```
hdbsql -n localhost:39613 -u SYSTEM -p MyPWD1234 "\s"
```

The above command returns the following result:

```
host          : localhost:39613
sid           : H4C
dbname         : SYSTEMDB
user          : SYSTEM
kernel version: 2.00.054.00.1611906357
SQLDBC version: libSQLDBCHDB 2.07.021.1611351107
autocommit    : ON
locale        : en_US.UTF-8
input encoding: UTF8
sql port      : linux-bj73:39613
```

Run the following command all on one line to execute the SELECT statement in command mode with simultaneous database logon:

```
hdbsql -n localhost -i 1 -u USER1 -p Password123
"SELECT CNO,TITLE,FIRSTNAME,NAME,ZIP FROM HOTEL.CUSTOMER"
```

The above command returns the following result:

```
CNO | TITLE | FIRSTNAME | NAME | ZIP
----+-------+-----------+------+------
3000 | Mrs | Jenny | Porter | 10580
3100 | Mr | Peter | Brown | 48226
3200 | Company | ? | Datasoft | 90018
3300 | Mrs | Rose | Brian | 75243
```

```
3400 | Mrs | Mary | Griffith | 20005
3500 | Mr | Martin | Randolph | 60615
3600 | Mrs | Sally | Smith | 75243
3700 | Mr | Mike | Jackson | 45211
3800 | Mrs | Rita | Doe | 97213
3900 | Mr | George | Howe | 75243
4000 | Mr | Frank | Miller | 95054
4100 | Mrs | Susan | Baker | 90018
4200 | Mr | Joseph | Peters | 92714
4300 | Company | ? | TOOLware | 20019
4400 | Mr | Antony | Jenkins | 20903
(15 rows selected) * Ok
```

Run multiple commands imported from a batch file in command mode:

```
hdbsql [<options>] -I CITES
```

The file contains the following statements for execution:

```
CREATE TABLE city(
    zip NCHAR (5) PRIMARY KEY,
    name NCHAR (20),
    state NCHAR (2));
CREATE TABLE customer(
    cno INTEGER PRIMARY KEY,
    title NCHAR (7),
    firstname NCHAR (10),
    name NCHAR (10),
    zip NCHAR (5),
    address NCHAR (25));
\dc city;
\dt customer;
SELECT DISTINCT name FROM city;
\sstats;
COMMIT;
```

10.4 Run Long Commands in Multiple-Line Mode

Multiple-line mode enables you to enter long commands, for example a long SQL statement on several lines. SAP HANA HDBSQL stores multiple-line commands in an internal command buffer.

Prerequisites

To run some commands, you must be logged on to the database.

Procedure

1. Activate multiple-line mode by running one of the following commands:

- Call option: `hdbsql [<options>] -m`
- SAP HANA HDBSQL command: `\mu ON`

2. Enter the command.

   To start a new line, press **Enter**.

3. Run the command in one of the following ways:

   - Close the last line of the command by entering a semicolon and pressing **Enter**.
   - SAP HANA HDBSQL command: `\g`.

Example

1. Log onto the SAP HANA database as user MONA with the password RED by running the following command: `hdbsql -n localhost -i 1 -u MONA,RED`
2. Activate multiple line mode by running the following command: `\mu ON`
3. Enter a multiple-line SQL statement:

```
SELECT ROUND(SUM("M")/1024/1024/1024,2) AS "Peak Used Memory GB" FROM
(SELECT SUM(CODE_SIZE+SHARED_MEMORY_ALLOCATED_SIZE) AS "M" FROM
SYS.M_SERVICE_MEMORY UNION SELECT SUM(INCLUSIVE_PEAK_ALLOCATION_SIZE) AS
"M" FROM M_HEAP_MEMORY_RESET WHERE DEPTH = 0)
```

4. Execute the SQL statement by entering the following command:`\g`

10.5 Edit Long Commands in an External File

If you have entered a long command in SAP HANA HDBSQL in multiple-line mode, then you can change it later by editing the command buffer in an external file and then re-running it.

Prerequisites

You have already run the command.

Procedure

1. To export the contents of the command buffer to an external file, run the following command:

```
\e <[file]>
```

   You must enter the complete file path and file name. If you do not specify a file, then SAP HANA HDBSQL generates a temporary file.

The system opens the file in an editor. To determine which editor is used, SAP HANA HDBSQL evaluates the environment variables HDBSQL_EDITOR, EDITOR, and VISUAL in succession. If you have not set any of these environment variables, then the visual editor is used on Linux and UNIX. On Windows, the variable must be set to the full path of an .exe file. For more information about setting environment variables, see your operating system documentation.

2. Make the required changes to the file.
3. Save the file in the editor and then close the file and the editor.

Results

You have changed the contents of the command buffer and can now execute the changed command by running the command `\g`.

10.6 Redirect Results to a File

Redirect the result of one or more SAP HANA HDBSQL commands to a file.

Prerequisites

To redirect results to a file, you must be logged on to the database.

Procedure

1. Run the following command:

   ```
   \o <file>
   ```

   You must enter the full path of the file.
2. Run the command whose result is being redirected to the file.
   To run multiple commands in succession, press `Enter` after each command.
3. To stop redirection to a file, run the following command: `\o`.

Example

To export a list of all schemas and all entries in the table HOTEL.CUSTOMER to an external file, perform the following steps.

1. Log onto the SAP HANA database as user MONA with the password RED by running the following command:

```
hdbsql -n localhost -i 1 -u MONA, RED
```

2. Create the file `c:\tmp\redirected.txt` then redirect the command result(s) to this file by running the following command:

```
\o c:\tmp\redirected.txt
```

3. Request information about all schemas by running the following command: `\ds`
4. Select all rows in the table HOTEL.CUSTOMER by executing the following statement:

```
SELECT * FROM HOTEL.CUSTOMER
```

5. Stop redirection to the file by running `\o`.

The `redirected.txt` file now contains the following content:

```
| Schema | Owner name |
| ------ | ---------------- |
| MDX_TE | SYSTEM |
| SECURI | SECURITY1 |
| SOP_PL | SYSTEM |
| SYS | SYS |
| SYSTEM | SYSTEM |
| _SYS_B | _SYS_REPO |
| _SYS_B | _SYS_REPO |
| _SYS_R | _SYS_REPO |
| _SYS_S | _SYS_STATISTICS |
| CNO | TITLE | FIRSTNAME | NAME | ZIP | ADDRESS|
| ------ | ------- | ---------- | ---------- | ----- | -------|
| 3200 | Company | ? | Datasoft | 90018 | 486 Maple Str.|
| 3400 | Mrs | Mary | Griffith | 20005 | 3401 Elder Lane|
| 3500 | Mr | Martin | Randolph | 60615 | 340 MAIN STREET, #7|
| 3600 | Mrs | Sally | Smith | 75243 | 250 Curtis Street|
| 3700 | Mr | Mike | Jackson | 45211 | 133 BROADWAY APT. 1|
| 3900 | Mr | George | Howe | 75243 | 111 B Parkway, #23|
| 4000 | Mr | Frank | Miller | 95054 | 27 5th Str., 76|
| 4400 | Mr | Antony | Jenkins | 20903 | 55 A Parkway, #15|
```

10.7 Substitution Variables

Substitution variables are processed by SAP HANA HDBSQL before the query that includes them is sent to the server.

Identifying Variables

Substitution variables are identified by either the prefix & or the prefix &&. If a variable is prefixed by &, then you are prompted to define the variable. If the variable is prefixed by &&, then you are prompted to specify a new value only if the variable has not already been defined.

You can also use the *[prompt]* and *[noprompt]* parameters of the -V option to control whether SAP HANA HDBSQL prompts you to define variables that it encounters in the script.

Rules for Defining Variables on the Command Line

Quoting and Spacing

Variable definitions can be non-quoted or quoted. Variable values that contain spaces must be single and double-quoted on the command line, and single-quoted in a SQL script file.

Single quotes on the command line must be escaped.

Non-quoted variable definitions cannot have spaces between the equal sign (=) and the variable name or definition, or between a comma and a variable definition.

The following example is not quoted because the variable definition contains no spaces and there are no spaces between the equal sign and the variable name and variable definition:

```
<variable-name>=<variable-value-with-no-spaces>
```

The following example is single and double-quoted because the variable definition contains spaces and is defined on the command line:

```
"<variable-name> = '<variable-value-with-spaces>'"
```

The following example is single-quoted only because the variable value contains spaces and is defined in a SQL script file:

```
<variable-name> = '<variable-value-with-spaces>'
```

Escape Characters

The following characters: a-z, A-Z, 0-9, |, and _ are allowed in variable names.

On the command line, variable values that contain quotes and backslashes must be escaped by using a backslash (\) and the string must be quoted.

The following example shows a variable value defined on the command line with special characters escaped, single quotes so that SAP HANA HDBSQL can process a variable value that includes spaces, and double quotes for the command shell to process a variable value that includes spaces:

```
-V VAR_1=\'"Sample string with\\ , and backslash \\\\"\'
```

The variable definition above is parsed as follows:

```
'"Sample string with\ , and backslash \\"'
```

10.8 Implement HTTP Proxy Client Connections for SAP HANA HDBSQL (SAP HANA Cloud)

An HTTP proxy connection allows direct TCP or TLS connections to be connected via an HTTP proxy for SAP HANA HDBSQL.

Context

Use HTTP proxy connections when direct TCP connections to an external server are not possible.

Procedure

1. Specify the SAP HANA HDBSQL command with the *-proxyhost* and *-proxyport*, and *-proxyhttp* options.
2. (Optional) Specify the *-proxyuserid* and *-proxypassword* options if your HTTP proxy requires user authentication.

Example

Use SAP HANA HDBSQL to connect to an SAP HANA Cloud server via an HTTP proxy:

```
hdbsql -u USER1 -p Password123 -n 724b1417-d273-4115-
a1f0-7a857742eefe.hana.canary-eu10.hanacloud.ondemand.com:443 -proxyhttp
-proxyhost httpproxyhost -proxyport 8080
```

Related Information

SAP HANA HDBSQL Options [page 858]
Implement HTTP Proxy Client Connections [page 33]
HTTP Proxy Client Connections [page 32]

11 SAP HANA Calculation View API

The SAP HANA Calculation View API is a Java-based Application Programming Interface (API) which allows you to create SAP HANA calculation views programmatically.

This API allows you to read, modify, and save the XML definitions of these calculation views, which can then be activated with the SAP HANA Deployment Infrastructure (HDI).

> ⓘ Note
>
> This API can only be used for HDI-based calculation views. Repository-based views as well as attribute or analytical views are not supported.

The SAP HANA Calculation View API for SAP HANA Cloud reflects the feature set included in SAP HANA Cloud. The latest version of the SAP HANA Calculation View is always compatible with the latest version offered by SAP HANA Cloud.

This API uses XML version 1.0 as this is the version of XML that the SAP HANA calculation views use.

Related Information

SAP HANA Calculation View API Reference
SAP HANA Cloud Deployment Infrastructure (HDI) Reference

11.1 Install the SAP HANA Calculation View API

The SAP HANA Calculation View API is included in the SAP HANA clients package.

Prerequisites

- During run time a JRE (Java Runtime Environment) of version 1.6 or higher is required
- During development a JDK (Java Development Kit) of version 1.6 or higher is required

> ⓘ Note
>
> An SAP HANA installation is **not** required to use the SAP HANA Calculation View API.
>
> For more information about Java 11 and higher, see SAP Note 2869216.

Procedure

1. The SAP HANA Calculation View API is included in the SAP HANA client installation package and includes the following files:
   - HANACalculationViewAPI-Cloud-XXX.jar (The API for SAP HANA Cloud)
   - HANACalculationViewAPI-Cloud-XXX-javadoc.jar (Javadoc for the API for SAP HANA Cloud)
2. Add the jar file and the SAP HANA Cloud or jar file to your class path, build path, or module path. If you are using an IDE, you can also link to the related Javadoc jar file.

Related Information

SAP Note 2869216
Getting Started with Calculation Views [page 880]

11.2 Getting Started with Calculation Views

Use calculation views to display detailed SAP HANA database information.

To load an existing Calculation View use the following code:

```
ScenarioRoot root = new ScenarioRoot():
root.loadWithXSDValidation(<xml>);
```

Create the calculation view manually by providing all the necessary elements, as illustrated below.

Define the input of the projection node:

```
Input projectionInput = new Input();
projectionInput.setNode("TAB");
AttributeMapping colInputMapping = new AttributeMapping();
colInputMapping.setTarget("COL");
colInputMapping.setSource("COL");
projectionInput.addMapping(colInputMapping);
```

Define the output of the projection node:

```
ViewAttributes projectionOutput = new ViewAttributes();
ViewAttribute colOutputAttribute = new ViewAttribute();
colOutputAttribute.setId("COL");
projectionOutput.addViewAttribute(colOutputAttribute);
```

Create the projection node:

```
CalculationViews calculationViews = new CalculationViews();
ProjectionView projectionView = new ProjectionView();
projectionView.setId("Projection");
projectionView.addInput(projectionInput);
projectionView.setViewAttributes(projectionOutput);
calculationViews.addCalculationView(projectionView);
```

Define the output of the calculation view:

```
Attributes calcViewAttributes = new Attributes();
Attribute attribute = new Attribute();
attribute.setOrder(1);
attribute.setId("COL");
attribute.setDisplayAttribute(false);
attribute.setSemanticType(SemanticType.EMPTY);
ColumnMapping colMapping = new ColumnMapping();
colMapping.setColumnObjectName("Projection");
colMapping.setColumnName("COL");
attribute.setKeyMapping(colMapping);
calcViewAttributes.addAttribute(attribute);
```

Define the semantic node:

```
MeasureGroup semanticNode = new MeasureGroup();
semanticNode.setId("Projection");
semanticNode.setAttributes(calcViewAttributes);
```

Create the calculation scenario:

```
CalculationScenario calculationScenario = new CalculationScenario();
calculationScenario.setId("MY_VIEW");
calculationScenario.setOutputViewType(CalculationViewType.PROJECTION);
calculationScenario.setDataSources(dataSources);
calculationScenario.setLogicalModel(semanticNode);
calculationScenario.setCalculationViews(calculationViews);
ScenarioRoot root = new ScenarioRoot();
root.setScenario(calculationScenario);
System.out.println(root.toXMLWithXSDValidation());
```

Use the following code to get the XML representation:

```
root.toXMLWithValidation();
```

Calculation Views Using SAP HANA Deployment Infrastucture

To read, write, and deploy calculation views from and to the database, you can use the SAP HANA Deployment Infrastructure (HDI). HDI offers an SQL-based API (described in the SAP HANA Deployment Infrastructure Reference), which can be utilized with the corresponding SQL driver for the programming language.

To use HDI, only the `diserver` service is needed, which is a native part of the SAP HANA, platform edition. No additional installation such as SAP HANA extended application services, advanced model (XS Advanced) is needed.

Related Information

SAP HANA Cloud Deployment Infrastructure Reference

11.3 API Object Methods

Use an API object method to manipulate an API object.

The following table describes the methods that are typically available for each API object. An API object is a Java object that is not part of the *com.sap.hana.cvapi.core.internal package*. For more information about API objects, see the Javadoc for the corresponding method.

Method	Description
toXML	Converts the object into its XML representation.
toXMLWithValidation	Converts the object into its XML representation including a validation.
set/add	Sets or updates a field by calling the corresponding set or add method.
remove/unset	Removes an entry from a list or sets a field to NULL.
get	Returns a field value.
has	Returns TRUE if a field has a value or if a list has any entry, otherwise this value is FALSE.
deepCopy	Returns a deep copy (create a copy of all references) of the actual object.
loadFromXML	Loads an XML object into the current object.
loadFromXMLWithValidation	Loads an XML object into the current object including a validation.
validate	Validates the current object.

The root object, `ScenarioRoot`, included in the package *com.sap.hana.cvapi.bimodel.calculation* has two additional methods, as described in the following table:

Method	Description
toXMLWithXSDValidation	Converts the object to its XML representation including a validation against the SAP HANA calculation view XML schema.
loadFromXMLWithValidation	Loads an XML into the current object including a validation against the SAP HANA calculation view XML schema.

11.4 Convert to Java Code

Convert each API class into its corresponding Java code.

Converting the API class into Java code means that the shortest possible Java code to create the current object state, including all referenced objects, is generated. The code only contains the constructor and setter calls. All objects are referenced with their fully qualified name, for example:

```
ModelClass c1 = new ModelClass();
c1.setFieldA(ModelEnum.FOO);
c1.setFieldB(new ModelClass());
c1.unsetA();
ToCodeProvider.toCode(c1);
```

The above code generates the following:

```
package1.ModelClass modelClass1 = new package1.ModelClass();
package1.ModelClass modelClass2 = new package1.ModelClass();
modelClass1.setFieldB (modelClass2);
```

> ⓘ Note
>
> The code generation is only intended to aid development and is not meant to be used in a productive environment to dynamically create code. It is recommended to check the generated code before use. The generated code is based on actual field values inside the used objects. Due to the nature of dynamic code generation, the field value might be used to inject malicious data into the code which could lead to a malfunction when the generated code is used.

11.5 Troubleshooting

Use the SAP HANA Calculation View API tracing function to troubleshoot issues.

You are responsible for protecting the generated trace files and for deleting them if they are not needed anymore.

To turn the tracing on, two method calls are needed. The first one sets the output file, the second one turns the tracing on.

Set the log file:

```
TraceConfiguration.configure(file);
```

Turn tracing on:

```
TraceConfiguration.turnOn();
```

To turn off tracing, run the following:

```
TraceConfiguration.turnOff();
```

The trace collects all modifications on all model objects. In addition, the trace operation also writes debug traces for the XML loading and generation. In the event of an error, the error stack trace and the corresponding code (if possible) are written.

Validation

The SAP HANA Calculation View API does not connect to an SAP HANA database. As a result, the API cannot validate whether any requirements inside the database are met, for example, if data sources exist.

Thread Safety

Unless explicitly stated otherwise in the Javadoc, all methods are not thread safe. The only exceptions are operations which are thread safe according to the Java language design.

User-Customized Java Platform Actions

The SAP HANA Calculation View API uses actions like reflection, logging, and file access which can be blocked by a security manager. Users are advised to configure their security manager (if one is used) in a way that allows all actions used by this API. Additionally, this API uses the Java platform default XML parser, transformer, and validator. If these tools are replaced with alternatives, the user is responsible for ensuring that the alternative implementation handles XML 1.0 and XSD 1.0 correctly.

11.6 Limitations and Unsupported Use

The SAP HANA Calculation View API comes with certain limitations.

The following lists the limitations of this API as well as unsupported use cases:

- Any access to a non-public method is not supported.
- All classes inside com.sap.hana.cvapi.core.internal are internal and may be changed or removed without notice.
- The generated XML cannot be used inside the SAP HANA repository.
- Any XML that should be loaded must be HDI compatible. Any SAP HANA repository XML is not supported.
- Any access to non-class files (resources) is not permitted.

12 SAP HANA Machine Learning APIs

The SAP HANA client install includes the option to install machine learning APIs that allow you to access and query SAP HANA data.

12.1 R Machine Learning Client for SAP HANA

The R machine learning client for SAP HANA provides a set of client-side R functions for accessing and querying SAP HANA data and a set of functions for developing machine learning models.

To use this API, you must run the SAP HANA client install and select the *R machine learning client for SAP HANA* option.

Related Information

Security
R Machine Learning Client for SAP HANA

12.1.1 Configure the R Machine Learning Client for SAP HANA

Install the R package into the R runtime.

Prerequisites

- You have installed SAP HANA Platform 2.0 SPS 03 or later (only these version os the PAL are supported)
- You have run the SAP HANA client install and selected the *R machine learning client for SAP HANA* option.
- The following R packages must be installed by using the command command `install.packages('<package>')` before you install the `hana.ml.r` package:
  - R6
  - sets
  - RODBC
  - futile.logger
  - uuid

Procedure

1. Install the R package into the R runtime.

Install Option	Command
Command line	`R CMD INSTALL hana.ml.r-1.0.4.tar.gz`
R code	`install.packages(<package-path>, repos = NULL, type="source")` In the command, `<package-path>` is the full path to `.tar.gz`. For example: Microsoft Windows: `"C:\\Program Files\\SAP\\hdbclient\ \hana.ml.r-1.0.4.tar.gz"` Linux/ macOS: `"/usr/sap/hdbclient/ hana.ml.r-1.0.4.tar.gz"`

2. Import the package to the R code:

```
library(hana.ml.r)
```

Results

You can reference the `hanaml` library object in code.

Related Information

R Machine Learning Client for SAP HANA
Security
What is SAP HANA Automated Predictive Library?

12.2 Python Machine Learning Client for SAP HANA

The Python machine learning client for SAP HANA provides a set of client-side Python functions for accessing and querying SAP HANA data and a set of functions for developing machine learning models.

The Python machine learning client for SAP HANA uses machine learning functions, particularly the Automated Predictive Library (APL) and the Predictive Analysis Library (PAL). To use the Python machine

learning client for SAP HANA, choose the *Python machine learning client for SAP HANA* option when you run the SAP HANA client install.

For information about installing and configuring APL, including required roles, refer to *SAP HANA Automated Predictive Library (APL)*.

For information about installing and configuring PAL, including required roles, refer to *SAP HANA Predictive Analysis Library (PAL)*.

Related Information

Python Machine Learning Client for SAP HANA
SAP HANA Predictive Analysis Library (PAL)
SAP HANA Cloud, SAP HANA Database Predictive Analysis Library (PAL)
What is SAP HANA Automated Predictive Library?

12.2.1 Configure Python Machine Learning Client for SAP HANA

Configure the Python machine learning client for SAP HANA.

Prerequisites

- You have installed SAP HANA Cloud or SAP HANA Platform 2.0 SPS 03 or later (only these versions of the PAL are supported)
- You have Python 3.6, 3.7, or 3.8 installed.
- You have installed the hdbcli driver version 2.2.23 or later.
- You have the APL `AFL__SYS_AFL_AFLPAL_EXECUTE` or `AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION` security roles for the Automated Predictive Library (APL) module of the SAP HANA Client API for ML.
- You have installed SAP HANA APL 1811 or later. This requirement is only necessary for using the APL package.
- You have run the SAP HANA client install and selected the *Python machine learning client for SAP HANA* option. During the install of the Python machine learning client for SAP HANA, the Python environment is checked for dependent libraries. Either an Internet connection is available for a dynamic install of the required libraries for the Python machine learning client for SAP HANA, or the libraries must be pre-installed in the respective Python environment. The required libraries are:
  - numpy
  - pandas
  - matplotlib
  - pydot

Procedure

Use a pip install to install the Python machine learning client for SAP HANA into a Python environment:

```
pip install <zip-file>
```

Related Information

Python Machine Learning Client for SAP HANA
Python Application Programming [page 665]
Security
What is SAP HANA Automated Predictive Library?

13 Important Disclaimer for Features in SAP HANA

For information about the capabilities available for your license and installation scenario, refer to the Feature Scope Description for SAP HANA. Some SAP HANA features and capabilities mentioned in this document are not applicable in the SAP BTP, SAP HANA service context. In some cases, additional information about the SAP HANA service context is placed near the feature information to advise you.

For information about the capabilities available for your contract type and provisioning scenario, refer to the Feature Scope Description for SAP HANA Service for SAP BTP or Feature Scope Description for SAP HANA Cloud.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

© 2024 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP