TIBCO Data Virtualization®

Client Interfaces Guide

Version 8.5.0
Last Updated: November 9, 2021

Contents

Introduction to Accessing Your Data through Client Interfaces

This topic gives an overview of the client interfaces TDV supports and introduces the process for implementing client interfaces. Topics covered include:

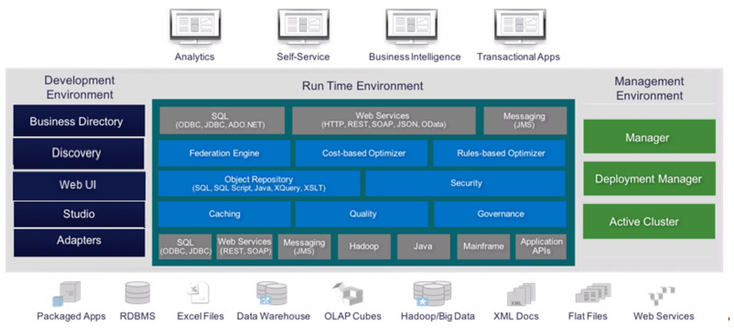
- About Client Interface Connections, page 12
- Connecting Client Applications to TDV Resources, page 13
- TDV Port Settings for Client Connections to TDV, page 15
- TDV Data Retrieval Tuning for Client Connections to TDV, page 17

About Client Interface Connections

This guide describes how to retrieve data through TDV using a client application. Client interfaces allow you to access data through TDV and consume it in your client applications. After you use TDV to define and publish views and other resource objects, you need to access that data so that you can perform data entry or complex data analysis on the data. Client applications can retrieve data from published resources in TDV through the following access mechanisms:

• JDBC	• ADO.NET
• REST	• ODBC
• OData	• SOAP

Before you can access and introspect underlying data sources, you must set up connections between your data sources and TDV. After using Studio to design and publish your virtual data resources, you need to set up the client interface connections to access those resources in TDV.



The exact methods that you use to connect to your published data resources in TDV vary depending on your system architecture, what the applications are, and how they need to connect to TDV. For example, your organization might need to write a Java application that uses ODBC to access the data through TDV. Another organization might need to create a Web application that uses SOAP to access the data through TDV.

Because the exact requirements of your organization are not known to us, we can only provide this guide as generalized information advising you on the steps that might be required. You need to analyze the advice in this guide and determine how to modify the steps to make them work for your organization.

Connecting Client Applications to TDV Resources

The general steps to configuring your client application to access data through TDV are described in this section.

To connect to data through TDV

1. Identify the underlying data sources, the virtual data resources that need to be built and published in TDV, and the client applications that will consume the TDV data resources.
2. Configure the data source connections between TDV and the underlying data sources.
3. Define and publish your data resources in TDV.

4. Determine how your client applications need to access the published resources in TDV (JDBC, ODBC, etc.) Refer to Driver Support, page 14 for the drivers supported.
5. Configure the connections between TDV and the client applications as described in one of the following sections:

Connector Type	See
JDBC	Connecting to TDV Server through JDBC, page 20
ODBC	Connecting to TDV Server through ODBC, page 62
SOAP	Connecting to TDV Server through SOAP, page 140
REST	Connecting to TDV Server through REST, page 142
OData	Connecting to TDV Server through OData, page 143
ADO.Net	Connecting to TDV Server through ADO.NET, page 146

6. Test your client application access to TDV.

Driver Support

Driver	Server Version	TDV Support
ODBC	iODBC Driver Manager v3.52.12 for AIX and Linux	Active
ODBC	Windows Driver Manager The Windows Driver Manager is part of your Windows Operating System. Refer to the section *Operating System Support for Server* in the *TDV Installation and Upgrade Guide* for a list of supported OS.	Active
ODBC	DataDirect Driver Manager v8.0 **Note**: DataDirect Driver Manager is supported on Linux and AIX platforms. There is no extra configuration needed for using TDV ODBC driver with DataDirect Driver Manager	Active

Driver	Server Version	TDV Support
ODBC	unixODBC Driver Manager(v2.3.1) **Note**: unixODBC Driver Manager is supported on Linux and AIX platforms. For using TDV ODBC driver with the unixODBC Driver Manager, set the environment variable TDV_ODBC_UODBC to TRUE for the applications that use TDV ODBC Driver. `set TDV_ODBC_UODBC=TRUE`	Active
JDBC	JRE v11 (csjdbc.jar) and conforms to JDBC API 4.0	Active
JDBC	JRE v1.8 (csjdbc8.jar) and conforms to JDBC API 4.0 **Example**: If you are running a client using JRE 8(also known as 1.8), you would include csjdbc8.jar in the CLASSPATH as shown below: "C:\Program Files\Java\jdk1.8.0_211\bin\java" -classpath .;.\csjdbc8.jar Test	Active
ADO.NET	ADO.NET (32-bit and 64-bit)	Active
Power BI Data Connector	20.0.7656	Active
SSIS	20.0.7668	Active
ADO.Net	ADO.Net 2021 Data Provider - v20.0.7656	Active

TDV Port Settings for Client Connections to TDV

The port settings for clients connecting to TDV using JDBC, ODBC, ADO.NET, and SSL are derived from the HTTP base port setting which is 9400 by default. The default settings for client connections are as follows:

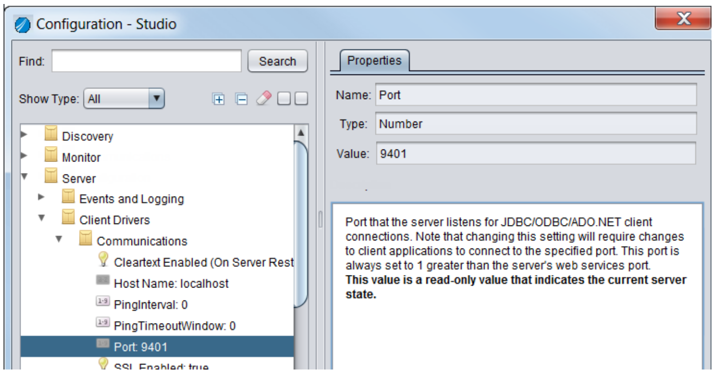
Connection Port	Default Port Numbers	Editable
HTTP base port	9400	Yes
JDBC, ODBC, and ADO.NET	9401 (base port + 1)	No
HTTP SSL	9402 (base port + 2)	No

Connection Port	Default Port Numbers	Editable
JDBC, ODBC, and ADO.NET SSL	9403 (base port + 3)	No

You can view the current base port, connection ports, and SSL security connection ports using Studio. Optionally, you can change the HTTP base port upon server restart which affects the other connection ports.

To view the JDBC, ODBC, and ADO.NET port value for client connections

1. Select Administration > Configuration from the Studio toolbar to open the Configuration panel.
2. Navigate to the Server > Client Drivers > Communications node. The Port value is the port used for JDBC, ODBC, and ADO.NET client connections. The SSL Port value is used for JDBC, ODBC, and ADO.NET client connections using SSL. Both values are derived from the HTTP base port and are read-only.



Name: Port

Type: Number

Value: 9401

Port that the server listens for JDBC/ODBC/ADO.NET client connections. Note that changing this setting will require changes to client applications to connect to the specified port. This port is always set to 1 greater than the server's web services port.
This value is a read-only value that indicates the current server state.

To change the HTTP base port and SSL port settings

1. Navigate to the Server > Web Services Interface > Communications > HTTP.
2. Select Port (Current). This value is the HTTP base port upon which all other ports are based.
3. Optionally, select Port (On Server Restart) and change the HTTP base port value. The SSL port for JDBC, ODBC, and ADO.NET will be the new value + 3.

   If you change the HTTP base port, consider the system impact. Remember that all of the other ports are derived from this value.
4. Click OK.

5. Restart your machine to make this change take effect.

TDV Data Retrieval Tuning for Client Connections to TDV

The way that data is retrieved from data sources and queued up in the TDV Server to provide data for your client applications can be tuned to optimize its retrieval. This method of retrieving data can approximate multithreaded data retrieval for client applications. This is especially beneficial for queries that return a large data set.

This feature improves the performance of queries by prefetching data from a data source and optimizing the data processing.

TDV can prefetch data for clients and store it on the TDV Server. The amount of data and the number of buffers used to store that data can be configured.

This tuning can be done using several TDV configuration parameters. The parameters work together to achieve results. Because each client application, query, and data source perform in different ways, it could take several attempts to determine the best settings for your exact circumstances.

Configuration Parameter	Description
DbChannel Prefetch Optimization	A boolean value that enables or disables this type of data retrieval tuning.
DbChannel Queue Size	An integer value between 0 and 100 that specifies the number of buffers that are allocated for prefetching data from a data source. It is recommended to use 10 as the starting value. If you observe an increase in memory usage this parameter can be used to control the amount of memory used to prefetch data. 0: No queuing or prefetch of the results done on the server. 1 – 100: • **If prefetch optimization is enabled:** This determines the max number of buffers a prefetch optimization can fill in on the server. The size of the buffer is determined by the fetchBytes, which can be set on the client. Default size of the fetchBytes is 128K. • **If prefetch optimization is disabled:** Is always interpreted as 1 and the buffer size is the lesser of fetchBytes or fetchRows.

Configuration Parameter	Description
DbChannel Zigzag String Optimization	Zigzag strings, which can save bytes over the wire, will be used in client communications if requested by the client. This setting can be used to ignore the client setting. If the setting is FALSE, the server will ignore the client setting, and the server will NOT send zigzag strings. If the setting is TRUE, the client can request zigzagStrings to be sent across the wire. It will not be necessary to restart server if this value changed.
DBChannel Flood Optimization	Setting the Flood Optimization option to True, will enable the Flood protocol to be used in client communications if requested by the client. The default value is True. **Note**: It will not be necessary to restart server if this value changed.
DBChannel Write Timeout Window	This option indicates the Timeout value when writing to the client, specified in seconds. Server closes a connection if it times out. The default value is 30 seconds.

To tune data retrieval

1. Select Administration > Configuration from the Studio toolbar to open the Configuration panel.
2. Search for DBChannel.
3. Set DbChannel Queue Size to 10 (as a starting value).
4. Select Apply.
5. Set DbChannel Prefetch Optimization to true.
6. Select Apply.
7. Select OK to save changes and exit the Configuration dialog.
8. If necessary set the value of fetchBytes for your client interface.

Connecting to TDV Server through JDBC

JDBC client applications can connect to the TDV Server to retrieve data from services managed, secured, and published by TDV-defined resources.

Installing JDBC Drivers

Make sure the JDBC driver is installed on the machine where you want to develop and run your JDBC client application that accesses data through the TDV Server. The TDV JDBC driver must be made available locally for each client that accesses TDV.

By default, TDV Server listens on port 9401 for JDBC connections, and the JDBC drivers are installed in subdirectories of the installation's root directory.

Note: Some clients might not be JDBC-4.0 compliant. An older version of the csjdbc.jar is available from Support to support older JDBC interfaces. If you have a previous release of TDV, you can use the csjdbc.jar file provided with that release.

To install the JDBC drivers to establish the connections between client

applications and the TDV Server

1. If TDV Server is running on a machine that has a Windows firewall, the firewall must be configured to allow JDBC clients to connect to the server through Studio.
2. Obtain and install the JDBC driver on the machine with the client application that accesses data through the TDV Server. Depending on the requirements of your organization, use your organizations copy of the JDBC driver or obtain the TDV JDBC driver from the TDV installer distribution. The TDV JDBC driver must be placed in an appropriate directory of any client application that connects to the TDV Server.
3. If you are using the AIX platform, see the *TDV Administration Guide* for information on "Configuring TDV for AIX Platforms."

Updating the JDBC Driver

When updating JDBC drivers, there is no need to update your client programs or connection strings.

To update the JDBC driver

1. Shut down all local applications using the JDBC driver.
2. Move the old csjdbc.jar file to an archive location.
3. Paste the newer file to the original csjdbc.jar location.
4. Restart your applications.
5. If you are using the AIX platform, see the *TDV Administration Guide* for information on "Configuring TDV for AIX Platforms".

Setting the Java CLASSPATH for the JDBC Driver

When using one of the TDV JDBC drivers with client applications written in Java, make sure that csjdbc.jar is available in your system's CLASSPATH. For the list of supported drivers, see Driver Support, page 14.

To set the CLASSPATH for JDBC driver

1. Make sure that the JDBC driver JAR file is available in your system's CLASSPATH.
2. Run the following command, depending on your operating system, to set the CLASSPATH:

OS	Command
Windows	`set CLASSPATH=%CLASSPATH%;<PATH>\csjdbc.jar`
UNIX	`export CLASSPATH=$CLASSPATH:<PATH>/csjdbc.jar`

<PATH> is the valid path where the JDBC driver is located.

3. Run your Java-based client application:

OS	Command
Windows	`java -classpath "%CLASSPATH%;<PATH>\csjdbc.jar" <CLIENT_JDBC_PROGRAM>`
UNIX	`java -classpath "$CLASSPATH:<PATH>/csjdbc.jar" <CLIENT_JDBC_PROGRAM>`

<CLIENT_JDBC_PROGRAM> is your client application that is written in Java and accesses data through TDV.

Setting Pass-Through Credentials for JDBC Clients

When data sources are configured to use pass-through login, the TDV Server session credentials are used by default to log in to the data source when no other credentials have been set by the JDBC client. Different data source credentials can be specified using the JDBC driver to negotiate data source access. When a data source is configured to use pass-through login, a data source credential establishes the connection and session. Credentials set with the JDBC driver are only valid for use with data sources that have been configured to use pass-through authentication, and connections created with them are only valid for the current JDBC client session.

Each named resource can be set with a username-password pair when creating connections with resources configured to use pass-through login. This can also be done for a generic "null data source" setting instance for unspecified resources.

The setDataSourceCredentials() method, registers the data source pass-through credentials for the current session for each data source specified. It can be called as many times as is necessary to set credentials on each pass-through login-enabled resource from which data is to be retrieved.

```
setDataSourceCredentials("<fullpath>","<username>", "<password>");
```

Variable	Description
<fullpath>	Provides the full path to the data source or can be NULL. If the path refers to a data source that the user does not have privileges to access, the program returns an error message.
<username>	Only one username-password pair can be set for a path.
<password>	Password to the data source.

Rules for Credentials Usage

The following rules govern the use of credentials:

- Connections and sessions with data sources are not created unless the client requests data from those resources.
- The connection and credentials are only valid for the current JDBC client session.
- For data sources configured to use pass-through, TDV does not re-use other connections, or sessions created by other users. Credentials set by the JDBC client are only available for the current client session. If the client connection is terminated abnormally, the connection is not returned to the pool for use by other clients.
- When the data source path matches a data source name for which credentials have been specified, those credentials are used to attempt data retrieval. To enhance security, any failure to connect or retrieve data with those credentials stops the retrieval process without attempts to use other credentials, even if suitable credentials were set on a NULL data source.
- If setDataSourceCredentials() is used to specify a user (e.g. 'userabc') normally, the data source will proceed to log in to the database server as this user. But the case in which the client has logged in to the TDV server as 'admin' is an exception. In this case, setDataSourceCredentials() is ignored and instead, TDV defaults to using the data source credentials to log in to the database server.

Note: When using pooled connections, it is recommended that JDBC clients clear any credentials that were set prior to returning the connection to the pool by calling the clearAllDataSourceCredentials (no arguments) method.

To set credential for pass-through login

1. Determine that you want to use pass-through login for your client application.
2. Read the Rules for Credentials Usage, page 23 to make sure that your situation qualifies for use of pass-through.
3. Add the connection URL to your client program. For example, for Java you might add:

```
String url = "jdbc:compositesw:dbapi@localhost:9401?"
                    +"domain=composite&dataSource=cdspt";
            String user = "compUser";
            String pass = "compPassword";
            // Load driver
            Class.forName("cs.jdbc.driver.CompositeDriver");
            // Create connection
            conn = DriverManager.getConnection(url, user, pass);
```

4. Add the setDataSourceCredentials method. For example add the following to your Java client application:

```
            ((cs.jdbc.driver.CompositeConnection)conn)
               .setDataSourceCredentials("/shared/sources/dsPassTh
ru",                 "dsUser", "dsPassword");
```

5. If you are using pooled connecAdd the clearAllDataSourceCredentials (no arguments) method to clear any credentials that were set prior to returning the connection to the pool
6. For more detailed sample code, see one of the following topics.

Reference	Description
Example 2: Set Data Source Credentials to Use Pass-through Data Sources, page 41	When data is requested from a data source enabled for pass-through login, the TDV Server first checks to see if the requesting client has credentials set for that named resource. If credentials have been set, they are passed through to establish a connection and session to get data from that source.

Reference	Description
Example 3: Setting Credentials for Use with Any Pass-through Data Source, page 42	If no credentials have been set on the data source from which data is required, the TDV Server checks to see if the setDataSourceCredentials() method was used to set credentials on the null data source (NULL). If setDataSourceCredentials()was used to set credentials on the null data source, those credentials are used to attempt data retrieval.
Example 4: Set TDV Server Credentials to Use Pass-through Data Sources, page 44	If credentials have not been set on the specifically named data source or on NULL, the TDV Server tries to access the data source using the same login credentials as those used to establish a connection with the TDV Server.

Connecting to TDV Server through TIBCO Spotfire

TIBCO Spotfire is a third-party tool. These instructions are included only as a guideline; your system and the steps necessary to configure it might vary from the test system that was used in this sample. You will need to refer to TIBCO Spotfire documentation and perform thorough testing of your system after completing the install and configuration.

This configuration to connect to TDV is based on Spotfire 3.1.

To connect to TDV Server through Spotfire

1. Check whether TDV is already installed with the Spotfire installation.
2. If TDV is installed, configure the connection to TDV server in Spotfire Information Designer using the TDV connection type.
3. If TDV is not installed yet, install TDV JDBC component on Spotfire box.
    a. Use TIBCO Spotfire Configuration Console to make sure that the Data Source Type is enabled. You might need to stop Spotfire services to enable TDV data type.
    b. Check Spotfire installation folders for csjdbc.jar. If this file does not exist locate it from the TDV installation zip and copy it to SPOTFIRE_INSTALL/../tomcat/webapps/spotfire/WEB-INF/lib folder.

    For later versions of Spotfire it could be recommended to put csjdbc.jar under ../tomcat/lib.

    c. Configure connection to TDV server in Spotfire Information Designer using TDV connection type.

4. Check connectivity.

Connecting to TDV Server through SQuirreL

SQuirreL is a third-party tool. These instructions are included only as a guideline; your system and the steps necessary to configure it might vary from the test system that was used in this sample. You will need to refer to SQuirreL documentation and perform thorough testing of your system after completing the install and configuration.

It is not necessary to have TDV installed on your computer to connect Squirrel to TDV. You can get a copy of the csjdbc.jar file from someone who has TDV installed. Save the csjdbc.jar file anywhere on your local file system, and provide its path to Squirrel so that Squirrel can find it.

To connect TDV and SQuirrel

1. Complete or review the instructions in the following sections:
   — Installing JDBC Drivers, page 20
   — Updating the JDBC Driver, page 21
   — Setting the Java CLASSPATH for the JDBC Driver, page 21
2. Note the location of your csjdbc.jar file.
3. Launch Squirrel.
4. Select the Drivers tab.
5. Click on the blue icon with a plus (+) sign or choose Drivers -> Add New Driver to create a new driver.
6. Type values for the following fields:

Field	For Example
Name	TDV Server
Example URL	jdbc:compositesw:dbapi@<server name>:9401?domain=<domain name>&dataSource=<TDV published datasource name>
Website URL	www.TIBCO.com (Users may use any website url)

7. Choose the Extra Class Path tab.
8. Click Add.

9. Type the full directory path location to your csjdbc.jar file.
10. Click Ok to get back the main screen.
11. Verify that TDV Server is added to the list of drivers. A message will be displayed when the driver is successfully created.
12. Select the Alias tab.
13. Click on the blue icon with a plus (+) sign to create a new alias.
14. Type values for the following fields:

Field	For Example
Name	DEV_TDV_Server<ver>
Driver	TDV Server
URL	jdbc:compositesw:dbapi@lctcvd0250:9401?domain=composite&dataSource=BBMS **Note**: The server, domain, datasource may vary according to your environment.
User Name	Use an existing account.
Password	

The URL points to a database called "BBMS." You might need to point to a different database.

15. Click Test and execute a SELECT query, to test your connection.

Defining a JDBC Client using a Connection URL

The following instruction are provided as guidelines only. You will need to determine exactly what your client programming environment requires.

This topic also includes the following:

- JDBC Driver Connection URL Properties, page 28

To create a client program

1. Create your client application and declare your connection URL, using the following syntax:

```
{TDV <version number>};Server=fully qualified
hostname;Port=9401;User=username;Password=password;domain=Composit
e domainname;dataSource=datasource name
```

   For example, for Java you might add:

```
String url = "jdbc:compositesw:dbapi@localhost:9401?"
                +"domain=composite&dataSource=cdspt";
            String user = "compUser";
            String pass = "compPassword";
            // Load driver
            Class.forName("cs.jdbc.driver.CompositeDriver");
            // Create connection
            conn = DriverManager.getConnection(url, user, pass);
```

   For other URL properties, see JDBC Driver Connection URL Properties, page 28.

2. Declare the username and password variables for use in the connection statement.

3. (Optional) Determine the JDBC driver name using one of the following methods, depending on platform type:

Platform	Location of Name
Windows	The Driver Name can be found from the Data Source tab of the JDBC Data Source Administrator.

4. (Optional) Write a small sample program that you can use to test the connection URL.

5. Create or modify your client program so that it includes the connection syntax. For example, you must include a statement similar to the following to establish the connection:

```
conn = DriverManager.getConnection(url, userName, password);
```

JDBC Driver Connection URL Properties

This table lists the names of properties that you can specify in the JDBC connection URL.

Non-alphanumeric characters within a NAME or VALUE must be URL-encoded.

JDBC Property Name	Description
alternatesecuritycredentials	Specifies an alternate security property value to the identity within the current session. This is used to allow the user passing security property to the data source. **Note**: You may get unexpected results when multiple requests are made on the same session or when multiple identities access the same session.
caseSensitive	Specifies case sensitivity in the request values. By default (false), requests are not case-sensitive.
commitFailure	Behavior if commit failed, possible values: rollback or bestEffort.
commitInterrupt	Specifies the behavior if a commit is interrupted, possible values are: ignore, log, fail.
compensate	If enabled, compensation blocks will be run if the transaction rolls back. Possible values: disabled or enabled. Default value is disabled.
connectTimeout	Time-out for initial connection, in seconds. Use 0 (zero) for infinite time-out.
convertToLocalTimezone	When set to the default value (false), the client receives TIMESTAMP values directly from the published data source, without conversion to the client's local time zone. If the value from the data source had a time zone associated with a TIMESTAMP, that value is preserved. When set to true, TIMESTAMP values are converted to the client's time zone (as in some earlier releases of TDV).
disableClustering	When set to the default value (false), enables data views from system tables from individual TDV instances in a cluster.
enableTDVConnectionPool	By default the value of this property is set to false. If set to true, the driver will revert to previous behavior where the connections to TDV server are pooled.

JDBC Property Name	Description
enableTDVTimestamp	Set this property in the JDBC driver to get the correct hour value, regardless of the current time zone or DST setting on the host server. When this property is enabled: • ResultSet.getObject() returns a custom child class of java.sql.Timestamp, with unchanged timestamp. • ResultSet.getTime() returns a java.sql.Time object with unchanged time (hour, minute, second), but the time from epoch is changed. • ResultSet.getString() returns the unchanged timestamp text. • ResultSet.getTimestamp() returns java.sql.Timestamp with changed timestamp regardless of the value of this property. Notes: 1) This JDBC property is required only if current time setting enables the clock to automatically adjust for daylight saving time (DST), and timestamp in database is a DST transition time (for example, from 2010-03-14 02:00:00 to 2010-03-14 02:59:59). 2) This JDBC property only affects getObject(), getTime(), and getString() of class ResultSet.
enableFastExec	Values are true or false, and the default value is false. Results are processed and returned immediately (instead of a round trip) when a query is submitted, potentially improving performance of low latency queries.
enableFlood	Values are true or false. Default value is true. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.
enableReconnectOnError	Specifies cluster reconnection behavior.
encrypt	When set to true, automatically passes JDBC messages to the SSL port for processing with the TDV SSL Certificate. See “Web Services Security” in the *TDV User Guide*.

JDBC Property Name	Description
fetchBytes	Maximum number of rows to fetch for a batch based on batch size, in bytes. Setting fetchBytes to a very large number can cause an Out Of Memory error in the server. The value set for fetchBytes affects the memory used on the JDBC client and the TDV server, so the value should be set based on the heap size configured.
fetchRows	Maximum number of rows to fetch for a batch. Set to zero to return an unlimited number of rows.
ignoreTrailingSpace	Ignore trailing spaces at the end of values. Default: false.
kerberos.krb5.conf	Path to krb5.conf file.
locale	Value that defines the user's language and country.
nometadata	Blocks return of result-set metadata during query execution.
paramMode	Controls the behavior of OUT parameters for stored procedures: • normal—Report OUT parameters in procedure metadata as OUT parameters. • return—Report OUT parameters as return values. • omit—Omit OUT parameters from metadata. • omitCursors—Omit output cursors from metadata.
pingInterval	Maximum time to wait before sending a ping request while waiting for result from TDV, in seconds.
pingTimeoutWindow	The length of time the JDBC or ODBC client waits before closing a connection to the TDV server, after a ping to the TDV server has failed. The value of this parameter should be greater than or equal to the "PingInterval" parameter. If a ping sent to the TDV server fails, the ODBC or JDBC client continues to send pings to TDV to check status. If these client pings continue to fail after the TimeoutWindow has expired, the ODBC or JDBC client closes the connection to the TDV server and sends a message. While the TimeoutWindow has not expired, the ODBC or JDBC client connection stays open and continues to send pings to the TDV server waiting for a response. The default for this property is "0", which means the setting is not being used. If not set, the session timeout and or request timeout is used instead.

JDBC Property Name	Description
registerOutputCursors	• true—Bind or register output cursors as output parameters. • false—Do not bind or register output cursors as output parameters; instead, use SQLMoreResults or Statement.getMoreResults() to access the cursors. See Example of Calling Procedures, page 38.
requestTimeout	Time-out for query commands and other requests.
sessionTimeout	Session inactivity time-out, in seconds. Set to zero for infinite time-out.
sessionToken	Uses the JDBC URL to set a session token value for client authorization when using TDV with a client restricted license. Example: &sessionToken=<VALUE>
singleLogSize	Maximum log file size to saving to next log file, in M bytes.
stripDuplicates	Values are true or false. Default value is false. If true, the server will detect for duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire. This would potentially lead to data savings across the wire.
stripTrailingZeros	Determines whether decimal result values are to be returned with trailing zeros removed.
traceLevel	Valid values are off, fatal, error (this is the default), debug, warn, info, debug, and all. The valid values for client-side log settings are off, fatal, error (default), warn, info, debug, all, stdout. On UNIX-based platforms, the log file CsOdbcDebug.log is created in the directory specified by the environment variable COMPOSITE_HOME.
unsupportedMode	Valid values are silent, warn, or fail. The default value is fail. When set to silent, unsupported methods do nothing and return. When set to warn, the JDBC driver logs a warning message in the log file, Otherwise, the JDBC driver returns a SQL_ERROR when it encounters unsupported methods. See Unsupported JDBC Methods, page 52.
user_tokens	Authentication values that can be packaged for delivery. The URL can pass the user_tokens property to the server at the init command, in the form: "user_tokens=(" NAME "=" VALUE ("," NAME "=" VALUE)* ")"

JDBC Property Name	Description
validateRemoteCert	False (default): no certificate validation is performed before establishing a connection. Also by default, a placeholder certificate is installed; csjdbc.jar uses a default bundled truststore for validation, unless the client system truststore is present and configured. True: The TDV JDBC client initiates the validation handshake, using the TDV certificate a for password encryption. If validation fails, no connection is established. The TDV Server certificate is loaded from the file specified in the Truststore File Location configuration parameter. The Keystore Key Alias is used when it is configured. The TDV JDBC client driver uses the client system's truststore properties to validate the certificate: • javax.net.ssl.trustStore • javax.net.ssl.trustStorePassword • javax.net.ssl.trustStoreType The TDV Server certificate must be added to this client's truststore; otherwise, validation fails. The placeholder TDV certificate does not work after the client system truststore is enabled, unless it is added to the client truststore.
validateRemoteHostname	False (default): No host name validation is performed. True: The csjdbc.jar compares the value of host in JDBC URL with the subject CN (common name) value in the certificate received from the targeted TDV Server. If host name validation fails, the connection is not established.
AccessToken	The authorization tokens used for OAuth2 authentication. The tokens are represented in a specific format - <header>.<payload>.<signature>. Each of the parts of the token is in a JSON format. The access token is used in place of id/password credentials, with a limited lifetime & privileges.

JDBC Property Name	Description
AccessTokenType	The type of the AccessToken. JWT (JSON Web Token) is the default supported format. JWT token is a self-contained JSON form and ideal for federation.

Examples

- Example Java JDBC Client Application Code, page 34
- Examples of Accessing Data through JDBC, page 39

Example Java JDBC Client Application Code

This section provides a sample template for using the JDBC driver in Java code. You must provide appropriate values for ip, datasource, userName, password, and the SQL statement.

```
import java.sql.*;
class JdbcSample
{
    public static void main(String args[])
    {
        if (args.length != 7) {
           System.err.println("usage : prog <datasource name> <host 
name> <port> <user> <password> <domain name> \"<sql statement>\"");
            System.exit(1);
        }

        String datasource = args[0]; // datasource_name
        String ip = args[1]; // IP or host name of TDV Server
        // port of TDV Server dbapi service
        int port = 0;
        try {
            port = Integer.parseInt(args[2]);
        } catch (Exception e) {
            port = 9401;
        }

        String userName = args[3];
        String password = args[4];
        String domain = args[5];
        String url = null;

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            Class.forName("cs.jdbc.driver.CompositeDriver");

            url = "jdbc:compositesw:dbapi@" + ip + ":" + port +
"?domain=" +
                domain + "&dataSource=" + datasource;

            conn = DriverManager.getConnection(url, userName,
password);

((cs.jdbc.driver.CompositeConnection)conn).clearAllDataSourceCrede
ntials();
((cs.jdbc.driver.CompositeConnection)conn)
.setDataSourceCredentials(<datasourcename>,user,password);
stmt = conn.createStatement();
boolean isNotUpdate = stmt.execute(args[6]);
            int rows = 0;

            // return type is a result set
            if (isNotUpdate == true) {
                rs = stmt.getResultSet();

                if (rs == null) {
                    throw new SQLException("sql=`"+args[6]+"` did
not generate a result set");
                }
                rsmd = rs.getMetaData();

                int columns = rsmd.getColumnCount();
                System.out.println("column count = " + columns);

                rows = 1;
                int type = 0;

                while (rs.next()) {
                    System.out.print("row = `" + rows + "`  ");
                    for (int i=1; i <= columns; i++) {
                        type = rsmd.getColumnType(i);
                        switch (type) {
                            case Types.INTEGER:
                              System.out.print(" col[" + i + "]=`"
+ rs.getInt(i) + "` ");
                                break;

                            case Types.SMALLINT:
                              System.out.print(" col[" + i + "]=`"
+ rs.getShort(i) + "` ");
                                break;

                            case Types.TINYINT:
                              System.out.print(" col[" + i + "]=`"
+ rs.getByte(i) + "` ");
                                break;

                            case Types.BIGINT:
                              System.out.print(" col[" + i + "]=`"
+ rs.getLong(i) + "` ");
                                break;

                            case Types.FLOAT:
                              System.out.print(" col[" + i + "]=`"
+ rs.getFloat(i) + "` ");
                                break;

                            case Types.REAL:
                              System.out.print(" col[" + i + "]=`"
+ rs.getFloat(i) + "` ");
                                break;

                            case Types.DECIMAL:
                              System.out.print(" col[" + i + "]=`"
+ rs.getFloat(i) + "` ");
                                break;

                            case Types.DOUBLE:
                              System.out.print(" col[" + i + "]=`"
+ rs.getDouble(i) + "` ");
                                break;

                            case Types.NUMERIC:
                              System.out.print(" col[" + i + "]=`"
+ rs.getFloat(i) + "` ");
                                break;

                            case Types.CHAR:
                              System.out.print(" col[" + i + "]=`"
+ rs.getString(i) + "` ");
                                break;

                            case Types.VARCHAR:
                              System.out.print(" col[" + i + "]=`"
+ rs.getString(i) + "` ");
                                break;
                            case Types.LONGVARCHAR:
                              System.out.print(" col[" + i + "]=`"
+ rs.getString(i) + "` ");
                                break;

                            case Types.DATE:
                              System.out.print(" col[" + i + "]=`"
+ rs.getDate(i) + "` ");
                                break;

                            case Types.TIME:
                              System.out.print(" col[" + i + "]=`"
+ rs.getTime(i) + "` ");
                                break;

                            case Types.TIMESTAMP:
                              System.out.print(" col[" + i + "]=`"
+ rs.getTimestamp(i) + "` ");
                                break;

                            case Types.BOOLEAN:
                              System.out.print(" col[" + i + "]=`"
+ rs.getBoolean(i) + "` ");
                                break;

                            default:
                              System.out.print(" col[" + i + "]=`"
+ rs.getString(i) + "` ");
                                break;
                        }
                    }

                    System.out.println("\n");
                    rows++;
                }

                rs.close();
            } else {
              // return type is not a result set
              rows = stmt.getUpdateCount();
              System.out.println("sql=`"+args[4]+"` affected " +
rows + " row(s)");
            }

            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
            if (rs != null)   {
                try {
                    rs.close();
                } catch (SQLException ignore) { }
            }
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException ignore) { }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException ignore) { }
            }
        } finally {
            rs = null;
            stmt = null;
            conn = null;
        }
    }
}
```

Example of Calling Procedures

JDBC supports the getMoreResults method and the getResultSet method. The following pseudocode illustrates how to use those methods when registerOutputCursor is set to true.

The registerOutputCursors property might not be best for use in JDBC, ODBC, and ADO.NET client connections. TDV provides a standard way to call procedures with or without registerOutputCursors in JDBC, ODBC, and ADO.NET connections.

```
String query = "{call LookupProduct(?,?,?,?)}";
cst = conn.prepareCall(query);
cst.setInt(1, 3);
cst.registerOutParameter(2, Types.OTHER);
cst.setInt(3, 3);
cst.registerOutParameter(4, Types.OTHER);
cst.execute();
/**
 * Method 1:
 */
rs = (ResultSet) cst.getObject(2);
rs = (ResultSet) cst.getObject(4);

/**
* Method 2:
*
```

```
* rs = cst.getResultSet();
* if(cst.getMoreResults())
* rs = cst.getResultSet();
*/
while (rs.next())
;
conn.close();
```

The following pseudocode shows how to use getResultSet() and getMoreResults() whether or not registerOutputCursors is set to true.

```
String query = "{call LookupProduct(?,?)}";
cst = conn.prepareCall(query);
cst.setInt(1, 3);
cst.setInt(2, 4);
cst.execute();

rs = cst.getResultSet();
if (cst.getMoreResults())
rs = cst.getResultSet();

while (rs.next())
;
conn.close();
```

Examples of Accessing Data through JDBC

The TDV JDBC driver supports the following statement types:

- Examples of Accessing Data through JDBC Using Statements, page 40
- Examples of Accessing Data through JDBC Using Prepared Statements, page 45

To use these examples in your TDV environment

1. Supply values for the following:
   — Login credentials for accessing TDV Server
   — Login credentials for accessing the data source
   — SELECT statement
2. Set the CLASSPATH to csjdbc.jar, which contains the TDV JDBC driver.
3. Compile and run your code.

Examples of Accessing Data through JDBC Using Statements

This section contains examples to illustrate the use of statements and specifications for pass-through login credentials.

- Example 1: Submit a Select Statement Using the JDBC Driver, page 40
- Example 2: Set Data Source Credentials to Use Pass-through Data Sources, page 41
- Example 3: Setting Credentials for Use with Any Pass-through Data Source, page 42
- Example 4: Set TDV Server Credentials to Use Pass-through Data Sources, page 44

Example 1: Submit a Select Statement Using the JDBC Driver

This example demonstrates how to submit a SELECT statement to the TDV JDBC driver.

In this example, the data source does not require pass-through login credentials. Instead, the login credentials are compUser and compPassword.

```
import java.util.*;
import java.sql.*;

public class SelectExample {
public static void main(String[] arg) throws Exception {
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
String url = "jdbc:compositesw:dbapi@localhost:9401?"
+"domain=composite&dataSource=cds";
String user = "compUser";
String pass = "compPassword";
// Load driver
Class.forName("cs.jdbc.driver.CompositeDriver");
// Create connection
conn = DriverManager.getConnection(url, user, pass);
// Create statement
stmt = conn.createStatement();
// Execute statement
rs = stmt.executeQuery("SELECT * FROM catalog.schema.table");
// Get column count
ResultSetMetaData rsmd = rs.getMetaData();
int columns = rsmd.getColumnCount();
// Get results
while(rs.next()) {
for (int i=0; i<columns; i++) {
```

```
Object o = rs.getObject(i+1);
if (o == null) {
System.out.print("[NULL]");
} else {
System.out.print(o.toString());
}
System.out.print(" ");
}
System.out.println();
}
} finally {
if (rs != null) {
rs.close();
}
if (stmt != null) {
stmt.close();
}
if (conn != null) {
conn.close();
}
}
}
}
```

Example 2: Set Data Source Credentials to Use Pass-through Data Sources

This example illustrates how to submit a SELECT statement to a data source that requires pass-through credentials (dsUser, dsPassword). These data source login credentials are different from the ones used for accessing TDV Server (compUser, compPassword).

```
import java.util.*;
import java.sql.*;

public class multiPassThruWithTDVLogInCred {
    public static void main(String[] arg) throws Exception {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
           String url = "jdbc:compositesw:dbapi@localhost:9401?"
                +"domain=composite&dataSource=cdspt";
            String user = "compUser";
            String pass = "compPassword";
            // Load driver
            Class.forName("cs.jdbc.driver.CompositeDriver");
            // Create connection
            conn = DriverManager.getConnection(url, user, pass);
            ((cs.jdbc.driver.CompositeConnection)conn)
              .setDataSourceCredentials("/shared/sources/dsPassTh
ru",       "dsUser", "dsPassword");
            // Create statement
            stmt = conn.createStatement();
            // Execute statement
            rs = stmt.executeQuery("SELECT * FROM
catalog.schema.table");
            // Get column count
            ResultSetMetaData rsmd = rs.getMetaData();
            int columns = rsmd.getColumnCount();
            // Get results
            while(rs.next()) {
                for (int i=0; i<columns; i++) {
                    Object o = rs.getObject(i+1);
                    if (o == null) {
                        System.out.print("[NULL]");
                    } else {
                        System.out.print(o.toString());
                    }
                    System.out.print(" ");
                }
                System.out.println();
            }
        } finally {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (conn != null) {
                conn.close();
            }
        }
    }
}
```

Example 3: Setting Credentials for Use with Any Pass-through Data Source

In this example, the path to the data source is specified as NULL. When NULL is specified as the resource path, the credential is added to the session’s list of generic credentials for the user.

The program tries to connect with the data source using different credentials for the user, but connects only with a data source that has the specified user name and password. By not having to specify a resource path, the client can be ignorant of data source namespace, at the cost of having to try various login credentials to achieve a successful connection.

```
import java.util.*;
import java.sql.*;

public class multiPassThruWithNull {
    public static void main(String[] arg) throws Exception {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
           String url = "jdbc:compositesw:dbapi@localhost:9401?"
                +"domain=composite&dataSource=cdspt";
            String user = "compUser";
            String pass = "compPassword";
            // Load driver
            Class.forName("cs.jdbc.driver.CompositeDriver");
            // Create connection
            conn = DriverManager.getConnection(url, user, pass);
            ((cs.jdbc.driver.CompositeConnection)conn)
              .setDataSourceCredentials(NULL, "dsUser",
"dsPassword");
            // Create statement
            stmt = conn.createStatement();
            // Execute statement
            rs = stmt.executeQuery("SELECT * FROM
catalog.schema.table");
            // Get column count
            ResultSetMetaData rsmd = rs.getMetaData();
            int columns = rsmd.getColumnCount();
            // Get results
            while(rs.next()) {
                for (int i=0; i<columns; i++) {
                    Object o = rs.getObject(i+1);
                    if (o == null) {
                        System.out.print("[NULL]");
                    } else {
                        System.out.print(o.toString());
                    }
                    System.out.print(" ");
                }
                System.out.println();
            }
        } finally {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (conn != null) {
                conn.close();
            }
        }
    }
}
```

Example 4: Set TDV Server Credentials to Use Pass-through Data Sources

This example is similar to Example 2: Set Data Source Credentials to Use Pass-through Data Sources, page 41. The login credentials for accessing TDV Server are the same as those for accessing the data source.

```
import java.util.*;
import java.sql.*;

public class multiPassThruWithTDVLogInCred {
    public static void main(String[] arg) throws Exception {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
           String url = "jdbc:compositesw:dbapi@localhost:9401?"
                +"domain=composite&dataSource=cdspt";
            String user = "dsUser";
            String pass = "dsPassword";
            // Load driver
            Class.forName("cs.jdbc.driver.CompositeDriver");
            // Create connection
            conn = DriverManager.getConnection(url, user, pass);
            ((cs.jdbc.driver.CompositeConnection)conn)
               .setDataSourceCredentials("/shared/sources/dsPassTh
ru"    "dsUser", "dsPassword");
            // Create statement
            stmt = conn.createStatement();
            // Execute statement
            rs = stmt.executeQuery("SELECT * FROM
catalog.schema.table");
            // Get column count
            ResultSetMetaData rsmd = rs.getMetaData();
            int columns = rsmd.getColumnCount();
            // Get results
            while(rs.next()) {
                for (int i=0; i<columns; i++) {
                    Object o = rs.getObject(i+1);
                    if (o == null) {
                        System.out.print("[NULL]");
                    } else {
                        System.out.print(o.toString());
                    }
                    System.out.print(" ");
                }
                System.out.println();
            }
        } finally {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (conn != null) {
                conn.close();
            }
        }
    }
}
```

Examples of Accessing Data through JDBC Using Prepared Statements

A prepared statement is an object that contains an SQL statement, possibly with varying input parameters, that can be executed multiple times. Use a question mark (?) as a placeholder for a parameter within the SQL statement. After all placeholder parameters are set, the query is executed.

For further details on prepared statements, see references on JDBC API, and the information provided on JDBC at http://www.oracle.com/technetwork/java/javase/jdbc/index.html.

- Restrictions When Using Prepared Statement, page 45
- Prepared Statements Examples, page 46

Restrictions When Using Prepared Statement

The following rules apply when you submit a prepared statement to the TDV JDBC driver to access the server:

- You can create and use multiple prepared statements on one connection to the server.
- The server maintains a cache of prepared statements, some of which exist across multiple connections, so that when you create a prepared statement

that is already in the cache, the server does not need to recreate the query plan. The cache size is configurable, and access is through Manager. For details, see the *TDV Administration Guide*.

- The placeholder for a query parameter can be used anywhere a literal can be used.
- Prepared statements with placeholder parameters (?) cannot used with Netezza data ship because all variables must be resolved before submitting SQL to the data source. Federated queries with database-specific native functions must be able to push SQL directly to the data source, or the query fails.
- The DatabaseMetaData.getMetaData( ) method is not supported. However, you can get ResultSetMetaData by using ResultSet.getMetaData( ).

Prepared Statements Examples

The following examples show how to use prepared statements in TDV Server.

- Example 1: SELECT Statement, page 46
- Example 2: INSERT Statement, page 48
- Example 3: UPDATE Statement, page 49
- Example 4: DELETE Statement, page 51

Example 1: SELECT Statement

The following sample code demonstrates how to use a prepared statement that contains a SELECT statement. In this example, the SELECT statement queries the customers table and retrieves the required data under a certain condition, which initially uses the placeholder parameter ?. This example uses a for loop to set values for parameters in the prepared statement.

```
import java.sql.*;

public class PreparedStatementSample
{
  private static final String COMPOSITE_URL =

"jdbc:compositesw:dbapi@localhost:9401?domain=composite&dataSource
=cdbs";

  private static final String COMPOSITE_DRIVER =
    "cs.jdbc.driver.CompositeDriver";

  private static final String COMPOSITE_USER = "admin";
  private static final String COMPOSITE_PASSWORD = "admin";

  public static void main(String[] args) {
    try {
      Class.forName(COMPOSITE_DRIVER);
    } catch (ClassNotFoundException ex) {
      ex.printStackTrace();
      return;
    }

    try {
      execute();
    } catch (SQLException ex) {
      ex.printStackTrace();
      return;
    }
  }

  private static void execute()
    throws SQLException
  {
    Connection conn = DriverManager.getConnection(
      COMPOSITE_URL, COMPOSITE_USER, COMPOSITE_PASSWORD);

    PreparedStatement stmt = conn.prepareStatement(
      "SELECT * FROM products WHERE ProductID = ?”);
    for (int i = 1; i <= 5 ; i++) {
    stmt.setInt(1, i);
    ResultSet rs = stmt.executeQuery();
    System.out.println("Row " + i);
    printResultSet(rs);
    rs.close();
}
stmt.close();
    conn.close();
  }

  private static void printResultSet(ResultSet rs)
    throws SQLException
  {
    ResultSetMetaData metaData = rs.getMetaData();

    while (rs.next()) {
for (int i=1; i<=metaData.getColumnCount(); i++) {
        System.out.println("  Column " + i + " " +
metaData.getColumnName(i) +
                           " " + rs.getString(i));
      }
    }
  }
}
```

Example 2: INSERT Statement

The following sample code demonstrates the usage of a prepared statement that contains an INSERT statement. This example works like Example 1: SELECT Statement, page 46, except that here executeUpdate() is used instead of executeQuery() to execute the SQL, and the result set is the number of rows affected by the insert operation.

```
import java.sql.*;
import java.math.BigDecimal;

public class PreparedStatementInsert
{
  private static final String COMPOSITE_URL =

"jdbc:compositesw:dbapi@localhost:9401?domain=composite&dataSource
=tutorial";

  private static final String COMPOSITE_DRIVER =
    "cs.jdbc.driver.CompositeDriver";

  private static final String COMPOSITE_USER = "admin";
  private static final String COMPOSITE_PASSWORD = "admin";


  public static void main(String[] args) {
    try {
      Class.forName(COMPOSITE_DRIVER);
    } catch (ClassNotFoundException ex) {
      ex.printStackTrace();
      return;
    }

    try {
      execute();
    } catch (SQLException ex) {
      ex.printStackTrace();
      return;
    }
  }

  private static void execute()
    throws SQLException
  {
    Connection conn = DriverManager.getConnection(
      COMPOSITE_URL, COMPOSITE_USER, COMPOSITE_PASSWORD);
    PreparedStatement stmt = conn.prepareStatement(
      "INSERT INTO products (ProductID, ProductName, UnitPrice)" +
      "VALUES (?, ?, ?)");
    stmt.setInt(1, 50);
    stmt.setString(2, "new");
    stmt.setBigDecimal(3, new BigDecimal(50.00));

    int rowsInserted = stmt.executeUpdate();
    System.out.println("Rows inserted " + rowsInserted);

    stmt.close();
    conn.close();
  }

  private static void printResultSet(ResultSet rs)
    throws SQLException
  {
    ResultSetMetaData metaData = rs.getMetaData();
    int rowIndex = 0;
    while (rs.next()) {
      System.out.println("Row " + rowIndex++);
      for (int i=1; i<=metaData.getColumnCount(); i++) {
        System.out.println("  Column " + i + " " +
metaData.getColumnName(i) +
                           " " + rs.getString(i));
      }
    }
  }
}
```

Example 3: UPDATE Statement

The following sample code demonstrates the usage of a prepared statement that contains an UPDATE statement. This example works similar to Example 2: INSERT Statement, page 48. Here, the result set is the number of rows affected by the update operation.

```
import java.sql.*;
import java.math.BigDecimal;

public class PreparedStatementUpdate
{
  private static final String COMPOSITE_URL =

"jdbc:compositesw:dbapi@localhost:9401?domain=composite&dataSource
=tutorial";

  private static final String COMPOSITE_DRIVER =
    "cs.jdbc.driver.CompositeDriver";

  private static final String COMPOSITE_USER = "admin";
  private static final String COMPOSITE_PASSWORD = "admin";

  public static void main(String[] args) {
    try {
      Class.forName(COMPOSITE_DRIVER);
    } catch (ClassNotFoundException ex) {
      ex.printStackTrace();
      return;
    }

    try {
      execute();
    } catch (SQLException ex) {
      ex.printStackTrace();
      return;
    }
  }

  private static void execute()
    throws SQLException
  {
    Connection conn = DriverManager.getConnection(
      COMPOSITE_URL, COMPOSITE_USER, COMPOSITE_PASSWORD);

    PreparedStatement stmt = conn.prepareStatement(
      "UPDATE products SET ProductName = ? WHERE ProductID = ?");

    stmt.setString(1, "newProduct");
    stmt.setBigDecimal(2, new BigDecimal(50.00));

    int rowsUpdated = stmt.executeUpdate();
    System.out.println("Rows updated " + rowsUpdated);

    stmt.close();
    conn.close();
  }

  private static void printResultSet(ResultSet rs)
    throws SQLException
  {
    ResultSetMetaData metaData = rs.getMetaData();
    int rowIndex = 0;
    while (rs.next()) {
      System.out.println("Row " + rowIndex++);
      for (int i=1; i<=metaData.getColumnCount(); i++) {
          System.out.println("  Column " + i + " " " +
metaData.getColumnName(i) +
                           " " + rs.getString(i));
        }
      }
    }
}
```

Example 4: DELETE Statement

The following sample code demonstrates the usage of a prepared statement that contains a DELETE statement. This example works similar to Example 2: INSERT Statement, page 48. Here, the result set is the number of rows affected by the delete operation.

```
import java.sql.*;
import java.math.BigDecimal;

public class PreparedStatementDelete
{
  private static final String COMPOSITE_URL =
"jdbc:compositesw:dbapi@localhost:9401?domain=composite&dataSource
=tutorial";

  private static final String COMPOSITE_DRIVER =
    "cs.jdbc.driver.CompositeDriver";

  private static final String COMPOSITE_USER = "admin";
  private static final String COMPOSITE_PASSWORD = "admin";

  public static void main(String[] args) {
    try {
      Class.forName(COMPOSITE_DRIVER);
    } catch (ClassNotFoundException ex) {
      ex.printStackTrace();
      return;
    }

    try {
      execute();
    } catch (SQLException ex) {
      ex.printStackTrace();
      return;
    }
  }

  private static void execute()
    throws SQLException
  {
    Connection conn = DriverManager.getConnection(
      COMPOSITE_URL, COMPOSITE_USER, COMPOSITE_PASSWORD);

    PreparedStatement stmt = conn.prepareStatement(
      "DELETE FROM products WHERE ProductID = ?");
    stmt.setInt(1, 50);
    int rowsDeleted = stmt.executeUpdate();
    System.out.println("Rows deleted " + rowsDeleted);

    stmt.close();
    conn.close();
}
```

Tips From an Expert on Duplicate Schema Names in a Catalog

Sometimes the JDBC driver returns a schema name multiple times in a catalog. For certain JDBC clients like Squirrel or DB Visualizer, when exact names of catalogs and/or schemas are used more than once in different places in a single TDV service the JDBC driver returns the name multiple times.

This occurs when you a virtual database has one or more catalogs that contain a schema with the exact same name. The number of times the identical schema is returned depends on how many catalogs it has been created under.

This is a limitation associated with the JDBC clients.

Unsupported JDBC Methods

Unsupported JDBC methods can be handled in one of two ways:

- If you try to access a JDBC method that TDV Server does not support, the system throws a SQLException with the message, “The operation X is not supported.”
- If you prefer that the system not throw exceptions when you use unsupported methods, set unsupportedMode to silent in the JDBC connection URL, as in the following example:

```
jdbc:compositesw:dbapi@localhost:9401?domain=composite&dataSource=
examples&unsupportedMode=silent
```

CallableStatements (not supported)

Array	getArray(int parameterIndex)
Array	getArray(String parameterName)
BigDecimal	getBigDecimal(int parameterIndex, int scale)
Object	getObject(int parameterIndex, Map map)
Object	getObject(String parameterName, Map map)
Ref	getRef(int parameterIndex)
Ref	getRef(String parameterName)
URL	getURL(int parameterIndex)
URL	getURL(String parameterName)
void	registerOutParameter(int parameterIndex, int sqlType, int scale)
void	registerOutParameter(int parameterIndex, int sqlType, String typeName)
void	registerOutParameter(String parameterName, int sqlType, int scale)
void	registerOutParameter(String parameterName, int sqlType, String typeName)
void	setNull(String parameterName, int sqlType, String typeName)
	setObject(String parameterName, Object x, int targetSqlType, int scale)
void	setURL(String parameterName, URL x)

Connections (not supported)

Properties	getClientInfo()
String	nativeSQL(String sql)
PreparedStatement	prepareStatement(String sql, int autoGeneratedKeys)
PreparedStatement	prepareStatement(String sql, int[] columnIndexes)
PreparedStatement	prepareStatement(String sql, String[] columnNames)
void	releaseSavepoint(Savepoint savepoint)
void	setHoldability(int holdability)
void	setReadOnly(boolean readOnly)
Savepoint	setSavepoint()
	setSavepoint(String name)
void	setTypeMap(Map map)

DatabaseMetaData (not supported)

boolean	allProceduresAreCallable()
boolean	allTablesAreSelectable()
ResultSet	getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)
int	getSQLStateType()
ResultSet	getSuperTables(String catalog, String schemaPattern, String typeNamePattern)
ResultSet	getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)
boolean	supportsConvert(int fromType, int toType)

PreparedStatements (not supported)

void	addBatch()
boolean	execute(String sql)
int	executeUpdate(String sql)
	setArray(int i, Array x)
void	setNull(int paramIndex, int sqlType, String typeName)
void	setObject(int parameterIndex, Object x, int targetSqlType, int scale)
void	setRef(int i, Ref x)
void	setURL(int parameterIndex, URL x)

Blob (not supported)

java.io.OutputStream	setBinaryStream(long pos)
int	setBytes(long pos, byte[] bytes, int offset, int len)
int	setBytes(long pos, byte[] bytes)

Clob (not supported)

java.io.OutputStream	setAsciiStream(long pos)
java.io.Writer	setCharacterStream(long pos)
int	setString(long pos, String str, int offset, int len)
int	setString(long pos, String str)

ResultSet (not supported)

boolean	absolute(int row)
void	afterLast()
void	beforeFirst()
void	cancelRowUpdates()
void	deleteRow()
boolean	first()
Array	getArray(int i)
Array	getArray(String colName)
InputStream	getAsciiStream(int columnIndex)
BigDecimal	getBigDecimal(int columnIndex, int scale)
String	getCursorName()

Object	getObject(int i, Map map)
Ref	getRef(int i)
InputStream	getUnicodeStream(int columnIndex)
URL	getURL(int columnIndex)
URL	getURL(String columnName)
void	insertRow()
boolean	last()
void	moveToCurrentRow()
void	moveToInsertRow()
Object	Object getObject(String colName, Map map)
boolean	previous()
Ref	Ref getRef(String colName)
void	refreshRow()
boolean	relative(int rows)
void	setFetchDirection(int direction)
void	updateArray(int columnIndex, Array x)
void	updateArray(String columnName, Array x)
void	updateAsciiStream(int columnIndex, InputStream x, int length)
void	updateAsciiStream(String columnName, InputStream x, int length)
void	updateBigDecimal(int columnIndex, BigDecimal x)
void	updateBigDecimal(String columnName, BigDecimal x)
void	updateBinaryStream(int columnIndex, InputStream x, int length)

void	updateBinaryStream(String columnName, InputStream x, int length)
void	updateBlob(int columnIndex, Blob x)
void	updateBlob(String columnName, Blob x)
void	updateBoolean(int columnIndex, boolean x)
void	updateBoolean(String columnName, boolean x)
void	updateByte(int columnIndex, byte x)
void	updateByte(String columnName, byte x)
void	updateBytes(int columnIndex, byte x[])
void	updateBytes(String columnName, byte x[])
void	updateCharacterStream(int columnIndex, Reader x, int length)
void	updateCharacterStream(String columnName, Reader reader, int length)
void	updateClob(int columnIndex, Clob x)
void	updateClob(String columnName, Clob x)
void	updateDate(int columnIndex, java.sql.Date x)
void	updateDate(String columnName, java.sql.Date x)
void	updateDouble(int columnIndex, double x)
void	updateDouble(String columnName, double x)
void	updateFloat(int columnIndex, float x)
void	updateFloat(String columnName, float x)
void	updateInt(int columnIndex, int x)
void	updateInt(String columnName, int x)
void	updateLong(int columnIndex, long x)

void	updateLong(String columnName, long x)
void	updateNull(int columnIndex)
void	updateNull(String columnName)
void	updateObject(int columnIndex, Object x, int scale)
void	updateObject(int columnIndex, Object x)
void	updateObject(String columnName, Object x, int scale)
void	updateObject(String columnName, Object x)
void	updateRef(int columnIndex, Ref x)
void	updateRef(String columnName, Ref x)
void	updateRow()
void	updateShort(int columnIndex, short x)
void	updateShort(String columnName, short x)
void	updateString(int columnIndex, String x)
void	updateString(String columnName, String x)
void	updateTime(int columnIndex, java.sql.Time x)
void	updateTime(String columnName, java.sql.Time x)
void	updateTimestamp(int columnIndex, java.sql.Timestamp x)
void	updateTimestamp(String columnName, java.sql.Timestamp x)

Statements (not supported)

void	addBatch(String sql)
void	clearBatch()
void	clearWarnings()
boolean	execute(String sql, int autoGeneratedKeys)

boolean	execute(String sql, int columnIndexes[])
boolean	execute(String sql, String columnNames[])
int[]	executeBatch()
int	executeUpdate(String sql, int autoGeneratedKeys)
int	executeUpdate(String sql, int columnIndexes[])
int	executeUpdate(String sql, String columnNames[])
void	getWarnings()
	ResultSet getGeneratedKeys()
void	setCursorName(String name)
void	setEscapeProcessing(boolean enable)
	setFetchDirection(int direction)
void	setMaxFieldSize(int max)

Connecting to TDV Server through ODBC

Both 32-bit and 64-bit ODBC client applications can connect to the TDV Server to retrieve published data from services managed, secured, and published by TDV-defined resources.

TDV data services are accessible through industry-standard ODBC driver managers. How you configure an ODBC data source depends on the driver manager. ODBC access to the TDV Server requires that an ODBC driver manager for your specific operating system be installed on the client machine.

The TDV ODBC driver conforms to the ODBC 3.5 specification.

ODBC Driver Requirements

ODBC client applications require a 32-bit or a 64-bit TDV driver to connect with TDV. The following ODBC drivers are available:.

Windows ODBC drivers	UNIX ODBC drivers
32-bit with ODBC API 3.x	32-bit and 64-bit for AIX and AIX PowerPC
32-bit with ODBC API 2.5	
64-bit for Intel/AMD CPU with ODBC 3.5 API	

To consume TDV resources through Excel:

- Publish the resources to a catalog.
- Use a bit version of the ODBC driver that matches the bit version of your Excel software. For example, if you use a 64-bit version of Excel, you must use a 64-bit version of the ODBC driver.

Installing the ODBC Driver

The TDV ODBC installation executables and dynamically loaded libraries are provided in a Client Installation Package zip file that can be downloaded from https://edelivery.tibco.com/storefront/eval/tibco-data-virtualization/prod11801.html. For example, for TDV 8.0, the file is named TIB_tdv_drivers_8.0.0_all.zip. This zip file contains installation programs for connecting your ODBC client applications to a TDV Server on all supported platforms.

To install the ODBC driver, see:

- Installing the ODBC Client Driver on Windows, page 64
- Installing the ODBC Client Driver on UNIX, page 65
- Updating the ODBC Driver, page 69

To remove the ODBC client driver on Windows, see:

- Uninstalling the ODBC Client Driver on Windows, page 65
- Uninstalling the ODBC Client Driver on Unix, page 69

Installing the ODBC Client Driver on Windows

The computer that has an ODBC client application must have either a 32-bit or a 64-bit driver to connect with TDV.

To install the ODBC client driver on a Windows machine

1. If TDV Server is running on a machine that has a firewall, the firewall must be configured to allow ODBC clients to connect to the server through Studio.
2. Unzip the TDV Client Installation Package file. For example, for TDV 8.0, the file is named TIB_tdv_drivers_8.0.0_all.zip.

   This file is an alternative way to obtain all of the ODBC drivers. It is a separate download from the main TDV installer.
3. Locate and run the ODBC installer for your machine:

OS Type	Platform	Installer Application	Location
32-bit	ODBC 3.5 API	CsOdbcInstall<version>.exe	<TDV_install_dir>\apps\odbc\win\
	ODBC 2.5 API	CSOdbcinstall<version>_2x.exe	<TDV_install_dir>\apps\odbc\win\
64-bit	Intel/AMD ODBC 3.5 API	CsOdbcInstall<version>_x64.exe	<TDV_install_dir>\apps\odbc\win64\

   For example, to run the installer from the Windows CMD, type one of these three commands:

```
CsOdbcInstall1100.exe -install
CsOdbcInstall1100.exe install
CsOdbcInstall1100.exe start
```

   To run the installer in silent mode, execute the script from within the folder where it is saved using the -install or install option.

More information on using ODBC drivers to connect with the TDV Server is available in the *TDV Administration Guide*.

Uninstalling the ODBC Client Driver on Windows

To uninstall the ODBC client driver on Windows

1. Rerun the TDV ODBC installer from the command line using the uninstall command option.

OS Type	Platform	Installer Application
32-bit	ODBC 3.5 API	CsOdbcInstall<version>.exe -uninstall
	32bit ODBC 2.5 API	CSOdbcinstall<version>_2x.exe -uninstall
64-bit	Intel/AMD ODBC 3.5 API	CsOdbcInstall<version>_x64.exe -uninstall

Installing the ODBC Client Driver on UNIX

TDV ODBC drivers are available for 32-bit or 64-bit UNIX operating systems. You must install the correct version for your environment.

Creating a DSN is done through the configuration utility. The interactive utility is driverConfig. Use driverConfig to reconfigure the driver files (when the driver file location has changed), and create, edit, list, or delete DSN entries.

The following describes the tasks for using the ODBC driver on UNIX:

- Setting the ODBC Environment Variables on UNIX, page 65
- Creating a DSN with driverConfig on UNIX, page 67

Setting the ODBC Environment Variables on UNIX

For examples and instructions for how to set UNIX environment variables, refer to your favorite UNIX guidelines. A typical command might be:

setenv PATH "/bin:/usr/bin:/usr/sbin:ucb/bin"

To set the ODBC environment variables

1. Log into the installation machine as the same user that installed the TDV software.
2. Set the following environment variables:

Variable	Description and Values

COMPOSITE_HOME	This optional variable allows you to specify the location where the TDV ODBC driver is installed. This is the full path to the <TDV_ODBC_install_dir> for the TDV ODBC driver.
	
	If this configuration is not set, you can run driverConfig with an absolute or relative path, for example:
	`./home/release/apps/odbc/linux64/bin/driverConfig`
	`./odbc/linux64/bin/driverConfig`
	`./bin/driverConfig`
	
	If this variable is set to /home/release, then when you create a DSN, it goes to /home/release/apps/odbc/linux64/lib to find the so files.

ODBCINI	Full path to the configuration file odbc.ini. It is generated during creation of DSN configuration with driverConfig. The ODBCINI and ODBCINSTINI files do not exist yet and will be created during DSN creation in the next step. It should be: <TDV_install_dir>/odbc.ini **Note**: Edit this file, if you want to add any connection properties in addition to the ones specified while creating the DSN. See ODBC Driver Connection String Properties, page 77 for a list of connection properties. For example, if you want to enable SSL, edit the odbc.ini file and add an entry “encrypt=true”.
ODBCINSTINI	Full path to the ODBC drivers configuration file odbcinst.ini. It is generated during DSN configuration with driverConfig. The ODBCINI and ODBCINSTINI files do not exist yet and will be created during DSN creation in the next step. It should be: <TDV_install_dir>/odbcinst.ini
LD_LIBRARY_PATH	This is specific to Linux. This path refers to the location of the iODBC driver manager files. The default location is: <TDV_install_dir>\apps\odbc\<platformType>\lib If you already have this variable, add the additional path to the existing path definition.
LIBPATH	This is specific to AIX. This path refers to the location of the iODBC driver manager files. The default location is: <TDV_install_dir>\apps\odbc\<platformType>\lib

Variable	Description and Values
SHLIB_PATH	This is specific to HP-UX. This path refers to the location of the iODBC driver manager files. The default location is: <TDV_install_dir>\apps\odbc\<platformType>\lib

3. Add these variables to the users profile that will be accessing the ODBC driver.

Creating a DSN with driverConfig on UNIX

A DSN is the logical name that is used by ODBC to access data. You can use an ODBC DSN entry to store the connection string values externally, to minimize the complexity of the connection string that you must define in your program.

You can create a DSN using the configuration utility driverConfig. This configuration utility helps you to reconfigure the driver files and create, edit, list, or delete DSN entries. You can use it when the driver file location has changed or is to be changed after installation.

On UNIX platforms, SysV semaphores are used to synchronize the read and write operations, and they are never deleted by ODBC drivers. The ODBC driver might run into an error if it is unable to create a new one because the maximum SysV count has been reached.

You can clean up semaphores using the ipcrm command.

To create a DSN using driverConfig

1. Make sure that you have Read and Write permissions on the following files:

```
odbc.ini
odbcinst.ini
```

2. Locate driverConfig.

3. Run the utility using the following command:

```
driverConfig

For example:
./home/release/apps/odbc/linux64/bin/driverConfig
./odbc/linux64/bin/driverConfig
./bin/driverConfig
```

4. Supply driverConfig with responses to set configurations in the odbc.ini and odbcinst.ini.

The Usage instructions of driverConfig is given below:

```
Usage: driverConfig [options]
where options are:
[-help]
[-view]
[-deleteDSN <name>]
[-uninstallDriver]
[-installDriver <driver path>]
[-configDSN <DSN attributes>]
```

The table below describes the different options:

Option	Description
help	Show this information and exit
view	View configuration and DSNs on this system and exit
deleteDSN <name>	Delete the named DSN
uninstallDriver	Uninstall ODBC 3.5 64-bit Driver
installDriver <driver path>	Install ODBC 3.5 64-bit Driver
configDSN <DSN attributes>	Create a new ODBC 3.5 DSN, where DSN attributes are of the form: "DSN=test;host=localhost;port=9401;uid=user Id;pwd=password;domain=composite;datasource=ds;catalog=cat"

Note:

- If this utility is executed without any options then it will run in interactive mode using Terminal I/O
- All option names are case-insensitive.
- If -help or -view is one of the options used, then no other options that are included, will be executed.
- The utility accepts more than one option.

- The order of actions that will be executed is:

  -deletedsn

  -uninstallDriver

  -installDriver

  -configDSN

If the environment variable COMPOSITE_HOME is defined, then it is used for evaluating the default ODBC driver path. COMPOSITE_HOME is used for evaluation only during driver installation in interactive mode.

Uninstalling the ODBC Client Driver on Unix

To uninstall the ODBC client driver on Unix

Rerun the driverConfig from the command line using the uninstall command option:

```
driverConfig -uninstallDriver
```

Updating the ODBC Driver

If you need to upgrade the ODBC driver that you use to connect your client applications to the TDV Server, follow the steps in this section.

To update the ODBC driver

1. Shut down all local applications that use the ODBC driver.
2. Install the newer version of the driver.
3. Open ODBC Data Source Administrator > Drivers.
4. Check Version and Date to make sure the newer version of the driver is installed.
5. Review all client applications that access TDV data through this connection.
6. Update all the connection string information.

Preparing TDV Data Services for ODBC Client Connections

TDV does not require any special configuration for clients to connect with it using 32-bit or 64-bit ODBC drivers.

After you publish a resource, ODBC application clients can use the resources. It is recommended, though not required, that the data service have a catalog for use with the ODBC driver.

The TDV ODBC driver supports code pages for the language sets supported by the host operating system. For Windows server installations, TDV uses multibyte-to-widechar and widechar-to-multibyte system calls to perform conversions. For UNIX server installations, TDV uses the iconv library to perform conversions.

By default, TDV Server listens to port 9401 for ODBC connections. The ODBC port number is always one greater than the server's web services HTTP base port which by default, is 9400. So the ODBC default port number is 9401. If SSL is used (encrypt is set to true), the ODBC driver automatically adds 2 to the port value so that the 9403 port is used. To determine the actual ODBC port settings, see TDV Port Settings for Client Connections to TDV, page 15.

To support any client regardless of type

1. Create a Data Service data source and catalog.
2. Configure the ODBC client to use the 32-bit or 64-bit ODBC driver.
3. Use the published data service and catalog as targets.
4. Determine the actual ODBC port setting which is based on the HTTP base port setting for TDV. See TDV Port Settings for Client Connections to TDV, page 15 for how to determine this value.
5. For early releases of Oracle 11g, there is an error in DG4ODBC which causes queries that access a published TDV view to hang indefinitely if the query contains a numeric WHERE clause filter. To temporarily work around this issue execute the following command in you SQL console prior to executing any queries.

```
ALTER SESSION SET CURSOR_SHARING = EXACT;
```

   Or, upgrade your Oracle 11g instance to release 11.2.0.3 or later. Refer to Oracle bug 11858021 for more information.

Configuring Each Windows System Data Source Name

ODBC clients can use a configured data source name (DSN) for TDV to communicate with published TDV data services. A DSN can also be used to configure a 64-bit Windows client application to use either the TDV 64-bit or the TDV 32-bit ODBC driver.

Both 32-bit and 64-bit client applications can use the same System DSN to connect with TDV data services. However, if you configure a User DSN, the connection is available to only a single user on the local machine.

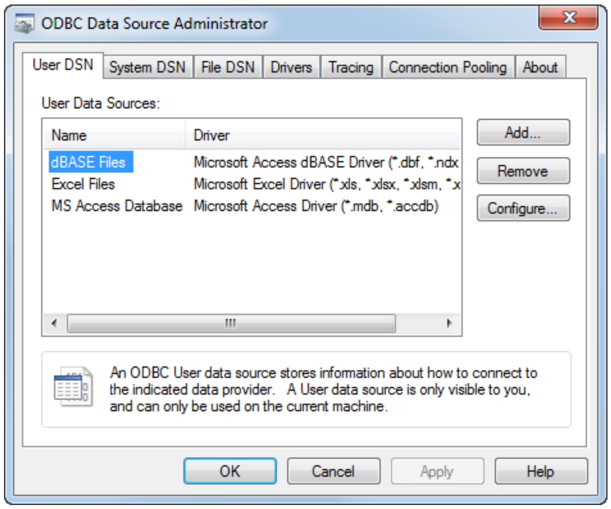
Adding a New System DSN

You can create a new System DSN to point ODBC clients to the appropriate data source published by TDV data services. If you plan to use Kerberos for integrated authentication, configure Kerberos following the instructions in the Kerberos topics in the *TDV Administration Guide* (in “Managing Security for TDV Resources” and “Configuring Kerberos Single Sign-On”).

To add a new system DSN

1. From the Windows Start menu, navigate to the ODBC Data Source Administrator window.

   For example, under Windows 10, select Start > Control Panel > Administrative Tools, and double-click Data Sources (ODBC) from the list on the right. The ODBC Data Source Administrator window opens.



Name	Driver
dBASE Files	Microsoft Access dBASE Driver (*.dbf, *.ndx
Excel Files	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.x
MS Access Database	Microsoft Access Driver (*.mdb, *.accdb)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

2. Select the System DSN tab and click the Add button.

   The Create New Data Source window opens.

3. In that window, highlight the TDV driver you previously installed and click Finish.

   The TDV <version> ODBC Driver Configuration window opens.

4. Configure the TDV Software ODBC driver by typing and selecting values as explained below.

Field	Description
DSN Name	A string for the client to use to address the data source.
Host	A DSN-designated name, or an IP octet, or localhost for a test client installed on the same computer as TDV.
Port	Number of the ODBC HTTP port (default 9401).

Field	Description
Encrypt	Check if you are using SSL authentication. If this is checked, the client will connect to the TDV server using the ODBC driver SSL port, (default 9403; the ODBC driver automatically uses base port + 2 for SSL client connections).
Integrated Authentication	Method for authenticating the ODBC connection: disabled (default), Kerberos, or NTLM.
Kerberos SPN	SPN for Kerberos to use to authenticate the ODBC connection. The text box is grayed out unless Kerberos is selected for Integrated Authentication.
User Name	TDV profile of the user connecting from the client. This should match a profile in either the Composite domain or a configured LDAP domain, unless anonymous or dynamic domain login is enabled. Custom Java Procedures (CJP) can use Windows ODBC client user names in the TDV runtime. When a client running on Windows makes an ODBC connection to TDV, the TDV ODBC driver uses a Windows API to report the Windows user name from the current session on the client. To retrieve the client user name within the CJP execution context /HOOK running on TDV, call ExecutionEnvironment.getProperty(loggedInUser). TDV does not use the client's Windows log-in user name for authentication.
Password	Password corresponding to user name.
Domain	Domain to which the user belongs.
Datasource	The TDV database name (the name of the database as it appears under Databases; no slashes) as published in the Databases node in the Studio resource tree. For example, if you have published resources under test (shown as /services/databases/test in the tool tip), type just "test" in this field.
Catalog	Each data source can publish one or more catalogs. Use the Refresh button when all other fields are specified, and select a catalog.
Locale/Code Page	The locale/code page used by this ODBC client. Leave this blank if the ODBC driver should use the system default locale. Otherwise, choose a different locale/code page. Refer to the section TDV Supported Encoding Standards, page 83 for a list of supported encoding standards by TDV.
Authentication Type	Indicates whether the authentication type is BASIC (using the user name/password) or OAUTH2 authentication (using access tokens). Choose OAuth2 if you will be using the authorization tokens.

Field	Description
Access Token	The authorization tokens used for OAuth2 authentication. The tokens are represented in a specific format - <header>.<payload>.<signature>. Each of the parts of the token is in a JSON format. The access token is used in place of id/password credentials, with a limited lifetime & privileges.
Token Type	The type of the AccessToken. JWT (JSON Web Token) is the default supported format. JWT token is a self-contained JSON form and ideal for federation.

5. Click the Test button to try the connection.

   If you do not get a message saying the connection test completed successfully, check the firewall and port setting to make sure the port is open for sending and retrieving messages.

6. Optionally, if you are using SSL security (you checked the Encrypt check box on the Basic tab), configure the Security tab for your Windows platform:

Field	Description
SSL Key ID	The subject (CN) of Windows private certificate. You can locate it using this procedure: 1. Run the certmgr.msc program to open the Windows Certificate Manager. 2. Under Personal/Certificates, double-click the private certificate to open the Certificate dialog. 3. Click the Details tab, then select the Subject field. Enter the Subject CN value in the SSL Key ID field. This is the only parameter you need to specify for Windows.
SSL Key Cert	The absolute path of a PEM file that contains the public key certificate for an SSL connection. (Optional)
SSL Key File	The absolute path of a PEM file that contains private key certificate for an SSL connection. This private key should match public key in SSL Key Cert. (Optional)
SSL CA Cert	The PEM file that contains the trusted CA certificates in PEM format. (Optional)

Field	Description
SSL CA Path	The absolute path of the directory that contains the trusted CA files in PEM format. On the Linux platform, the default value is "/etc/ssl/certs". The CA PEM file name in the CA path directory must equal the hash value for the CA PEM file name. (Optional) Note that on the Windows platform, the ODBC driver loads all CA certificates from the system store ROOT/CA/TRUST, so this parameter is not used.
Validate Remote Certification	Check to validate the remote certifications.
Validate Remote Hostname	Check to validate the remote hostname.

7. Click OK back through the windows to save the new System DSN.

With these configurations, both the TDV Server and the 32-bit or 64-bit ODBC client should be ready for use.

Note: SQL statements generated by ODBC clients must enclose reserved keywords in double-quotes when they are used as column aliases. For a consolidated list of reserved keywords, see the TDV *Reference Guide*. Edit an auto-generated MS-Query by renaming a column name with an alias.

Override the Configured Settings

A client connecting to a TDV Server through the ODBC driver can override the configured settings on a data source by adding parameters in the connection string. For example, the following connection string is valid with TDV Server:

```
DSN=<value>;UID=<value>;PWD=<value>;DOMAIN=<value>;HOST=<value>;PO
RT=<value>;DATASOURCE=<value>; CATALOG=<value>;
```

For these parameters:

- The original DSN value cannot be overridden.
- During creation of a DSN, you are prompted for a PWD entry.
- HOST is the host name where TDV Server is running.
- CATALOG is optional.

Defining an ODBC Client using a Connection String

It is possible to create client program and establish a connection to your data through TDV without having to define a DSN.

Note: The following instruction are guidelines only.

This topic also includes the following:

- ODBC Driver Connection String Properties, page 77
- C++ Example using the Connection String (DSN-less connection), page 95

To create a client program without defining a DSN connection

1. Create and declare your connection string, using the following syntax:

```
{Driver=<driver name>;Server=<fully qualified
hostname>;Port=<port>;User=<username>;Password=<password>;domain=<
domain name>;dataSource=<datasource name>
```

The following examples show how the syntax might be implemented in a C++ program.

Platform	Example
Windows	`SQLCHAR dsn[] = "Driver={TDV8.0};Server=localhost;Port=9401;User=admin;Password=admin;Domain=composite;dataSource=redwood;user=admin;password=admin;validateRemoteHostname=false;connectTimeout=3000;enableFastExec=false";`
Linux/UNIX	`SQLCHAR dsn[] = “Driver=composite70;Server=localhost;Port=9401;User=admin;Password=admin;domain=composite;dataSource=redwood;validateRemoteHostname=false;connectTimeout=3000;enableFastExec=false”;`

For other connection properties, see ODBC Driver Connection String Properties, page 77.

2. Declare the user name and password variables for the connection statement.

3. (Optional) Determine the ODBC driver name using one of the following methods.

Platform	Location of Name
Windows	The driver name can be found from the Drivers tab of the ODBC Data Source Administrator.
UNIX	Locate the driver name in the section name of the odbcinst.ini file; for example: composite70.

4. (Optional) Write a small sample program to test the connection string.
5. Create or modify your client program so that it includes the connection syntax. For example, you must include a statement similar to the following to establish the connection:

```
conn = DriverManager.getConnection(connection string, userName,
password);
```

ODBC Driver Connection String Properties

This table lists the names of properties that you can specify in the ODBC connection string.

ODBC Property	Description
alternatesecuritycredentials	Specifies an alternate security property value to the identity within the current session. This is used to allow the user passing security property to the data source. **Note**: You may get unexpected results when multiple requests are made on the same session or when multiple identities access the same session.
caseSensitive	Specifies case sensitivity in the request values. Values can be: • <EMPTY>(default) - the server settings will manage the case sensitivity • TRUE - requests are case-sensitive • FALSE - requests are not case-sensitive
commitFailure	Specifies the behavior if commit failed. Possible values are: rollback or bestEffort.

ODBC Property	Description
commitInterrupt	Specifies behavior if commit is interrupted.Possible values are: ignore, log, fail.
compensate	Specifies correcting behavior. If enabled, compensation blocks will be run if the transaction rolls back. Possible values: disabled or enabled. Default value is disabled.
connectTimeout	Time-out for initial connection, in seconds. Use 0 (zero) for infinite time-out.
currentLoggedInUserName	Current login user name.
dataSizeLimit	Specifies the maximum text column data size.
dataSource	Specifies the data source that is used for all connections.
domain	Specifies an identification string that defines a realm of administrative autonomy, authority, or control.
driver	The ODBC driver absolute path name.
dsn	ODBC DSN name.
enableFastExec	Valid values are true and false. The default value is false. Results are processed and returned immediately (instead of a round trip) when a query is submitted, potentially improving performance of low latency queries.
enableFlood	Values are true or false. Default value is True. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.
enableReconnectOnError	Specifies cluster reconnection behavior.
encrypt	If True, the client will connect to the TDV server using the ODBC driver SSL port.
fetchBytes	Maximum number of rows to fetch for a batch based on batch size, in bytes. Setting fetchBytes to a very large number can cause an Out Of Memory error in the server. The value set for fetchBytes affects the memory used on the client and the TDV server, so the value should be set based on the heap size configured. The default value is used if this property is set to zero.

ODBC Property	Description
fetchRows	Maximum number of rows to fetch for a batch. The default value is used if this property is set to zero.
host/server	TDV Server host name.
ignoreTrailingSpace	Ignore trailing spaces at the end of values. Values can be: • <EMPTY>(default) - the server settings will be used • TRUE - trailing spaces will not be ignored • FALSE - trailing spaces will not be ignored
locale	Value that defines the user's language and country. Refer to the section TDV Supported Encoding Standards, page 83 for a list of supported encoding standards by TDV.
nometadata	Blocks return of result-set metadata during query execution.
paramMode	Controls the behavior of OUT parameters for stored procedures: • normal—Report OUT parameters in procedure metadata as OUT parameters. • return—Report OUT parameters as return values. • omit—Omit OUT parameters from metadata. • omitCursors—Omit output cursors from metadata.
password/pwd	Specifies the password for the user name that you specify in the Username property. These values are used for your data source connection.
pingInterval	Maximum time to wait before sending a ping request while waiting for a result from TDV, in seconds.

ODBC Property	Description
pingTimeoutWindow	The length of time the JDBC or ODBC client waits before closing a connection to the TDV server, after a ping to the TDV server has failed. The value of this parameter should be greater than or equal to the "PingInterval" parameter. If a ping sent to the TDV server fails, the ODBC or JDBC client continues to send pings to TDV to check status. If these client pings continue to fail after the TimeoutWindow has expired, the ODBC or JDBC client closes the connection to the TDV server and sends a message. While the TimeoutWindow has not expired, the ODBC or JDBC client connection stays open and continues to send pings to the TDV server waiting for a response. The default for this property is 0, which means the setting is not being used.
port	TDV Server listening port.
registerOutputCursors	• true—Bind or register output cursors as output parameters. • false—Do not bind or register output cursors as output parameters; instead, use SQLMoreResults to access the cursors.
requestTimeout	Time-out for query commands and other requests
sessionTimeout	Session inactivity timeout, in seconds. Set to zero for infinite timeout.
singleLogSize	Maximum log file size to saving to next log file, in M bytes.
spn	Valid on Windows platform only, not useful on UNIX platforms. Kerberos SPN value, only useful if the SSO value equals Kerberos.
sso	Valid on Windows platform only, not useful on UNIX platforms. Single-sign-on type: ""/(Disabled), Kerberos or NTLM. The default value is "", which forces the ODBC client application to provide user and password information to connect.

ODBC Property	Description
sslKeyID	The subject (CN) of the Windows private certificate. You can locate this using this procedure: 1. Run the certmgr.msc program to open the Windows Certificate Manager. 2. Under Personal/Certificates, double-click the private certificate to open the Certificate dialog. 3. Click the Details tab, then select the Subject field. Enter the Subject CN value in the sslKeyID field. This is the only parameter you need to specify for Windows.
sslKeyCert	The absolute path of a PEM file that contains the public key certificate for an SSL connection. (Optional)
sslKeyFile	The absolute path of a PEM file that contains private key certificate for an SSL connection. This private key should match public key in SSL Key Cert. (Optional)
sslCACert	The PEM file that contains the trusted CA certificates in PEM format. (Optional) The properties sslCACert and sslCAPath should not be used on windows. This is because Windows has its own Trust Store in the operating system. Customers are advised to add non-standard CA certificates(example self-signed certificate(s)) in Windows TrustStore.
sslCAPath	The absolute path of the directory that contains the trusted CA files in PEM format. On the Linux platform, the default value is "/etc/ssl/certs". The CA PEM file name in the CA path directory must equal the hash value for the CA PEM file name. (Optional) Note that on the Windows platform, the ODBC driver loads all CA certificates from the system Truststore, so this parameter is not used.
stripDuplicates	Values are true or false. Default value is false. If true, the server will detect for duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire. This would potentially lead to data savings across the wire.
stripTrailingZeros	Determines whether decimal result values are to be returned with trailing zeros removed.

ODBC Property	Description
traceFile	Absolute path to the trace file.
traceFolder	Absolute directory to save trace file, the trace file name is "CsOdbcDebug_"+<DSN Name>+".log". the default folder is C:\ or $COMPOSITE_HOME
traceLevel	Valid values are off, fatal, error (this is the default), debug, warn, info, debug2, and all. The valid values for client-side log settings are off, fatal, error (default), warn, info, debug, all, debug2, stdout. On UNIX-based platforms, the log file CsOdbcDebug.log is created in the directory specified by the environment variable COMPOSITE_HOME.
unsupportedMode	Valid values are silent, warn, or fail. The default value is fail. When set to silent, unsupported methods do nothing and return. When set to warn, the JDBC driver logs a warning message in the log file, Otherwise, the JDBC driver returns a SQL_ERROR when it encounters unsupported methods.
user_tokens	Authentication values that can be packaged for delivery.
user/uid	Specifies the user name for connections to the data source.
validateRemoteCert	When true, the TDV client initiates handshake validation, validating the TDV certificate and using it for password encryption. If validation fails, no connection is established. When false (default), no certificate validation is performed prior to the establishment of a connection. The TDV Server certificate is loaded from the Truststore File Location set in the Studio Configuration panel. The Keystore Key Alias is used when it is configured for use. For more information, refer to “TDV Configuration Parameters” in the *TDV Administration Guide*. The TDV ODBC driver uses the system certification store to validate the certificate. The TDV Server certificate must be added to this client trust store or validation fails.

ODBC Property	Description
validateRemoteHostname	When true, the ODBC driver compares the value of host in the connection string with the subject CN (common name) value in the certificate received from the targeted TDV Server. If the host name validation fails, the connection is not established. When false (default), the host name validation is not performed.
authenticationType	Indicates whether the authentication type is BASIC (using the user name/password) or OAUTH2 authentication (using access tokens). Choose OAuth2 if you will be using the authorization tokens.
accessToken	The authorization tokens used for OAuth2 authentication. The tokens are represented in a specific format - <header>.<payload>.<signature>. Each of the parts of the token is in a JSON format. The access token is used in place of id/password credentials, with a limited lifetime & privileges.
accessTokenType	The type of the Access Token. JWT (JSON Web Token) is the default supported format. JWT token is a self-contained JSON form and ideal for federation.

TDV Supported Encoding Standards

This section lists the supported encoding standards for the ODBC Client Driver.

Windows

Use the encoding that is appropriate for your locale using the Locale option while configuring your DSN. The default value, if not specified, is the System default for the Windows OS.

windows874	shift_jis	gb2312	ks_c_56011987
big5	ibm1026	ibm01047	ibm01140
ibm01141	ibm01142	ibm01143	ibm01144
ibm01145	ibm01146	ibm01147	ibm01148

ibm01149	utf16	utf16le	utf16fffe
utf16be	windows1250	windows1251	windows1252
windows1253	windows1254	windows1255	windows1256
windows1257	windows1258	johab	macintosh
xmacjapanese	xmacchinesetrad	xmackorean	xmacarabic
xmachebrew	xmacgreek	xmaccyrillic	xmacchinesesimp
xmacromanian	xmacukrainian	xmacthai	xmacce
xmacicelandic	xmacturkish	xmaccroatian	utf32
utf32le	utf32be	xchinese_cns	xchinesecns
xcp20001	xchineseeten	xchinese_eten	xcp20003
xcp20004	xcp20005	xia5	xia5german
xia5swedish	xia5norwegian	usascii	xcp20261
xcp20269	ibm273	ibm277	ibm278
ibm280	ibm284	ibm285	ibm290
ibm297	ibm420	ibm423	ibm424
xebcdickoreanext ended	ibmthai	koi8r	ibm871
ibm880	ibm905	ibm00924	eucjp
xcp20936	xcp20949	cp1025	koi8u
iso88591	iso88592	iso88593	iso88594
iso88595	iso88596	iso88597	iso88598
iso88599	iso885913	iso885915	xeuropa
iso88598i	iso2022jp	csiso2022jp	iso2022jpx
iso2022kr	xcp50227	eucjp	euccn
euckr	hzgb2312	gb18030	gbk

xisciide	xisciibe	xisciita	xisciite
xisciias	xisciior	xisciika	xisciima
xisciigu	xisciipa	utf7	utf8

Unix

Use the encoding that is appropriate for your locale using the Locale option while configuring your DSN. The default value, if not specified, is "iso88591" for the Unix OS.

ascii	iso88591	iso88592	iso88593
iso88594	iso88595	iso88597	iso88599
iso885910	iso885913	iso885914	885915
iso885915	iso885916	koi8r	koi8u
koi8ru	cp1250	cp1251	cp1252
ibm1252	cp1253	cp1254	cp1257
cp850	cp866	macroman	maccentraleurope
maciceland	maccroatian	macromania	maccyrillic
macukraine	macgreek	macturkish	macintosh
iso88596	iso88598	cp1255	cp1256
cp862	machebrew	macarabic	eucjp
ibmeucjp	shift_jis	cp932	iso2022jp
iso2022jp2	iso2022jp1	euccn	ibmeuccn
hz	gbk	gb18030	euctw
ibmeuctw	big5	cp950	big5hkscs
iso2022cn	iso2022cnext	euckr	ibmeuckr
cp949	iso2022kr	johab	armscii8

georgianacadem y	georgianps	koi8t	tis620
cp874	macthai	mulelao1	cp1133
viscii	tcvn	cp1258	hproman8
nextstep	utf8	ucs2	ucs2be
ucs2le	ucs4	ucs4be	ucs4le
utf16	utf16be	utf16le	utf32
utf32be	utf32le	utf7	c99
java	ucs2internal	ucs4internal	char
wchar_t	cp437	cp737	cp775
cp852	cp853	cp855	cp857
cp858	cp860	cp861	cp863
cp865	cp869	cp1125	cp864
eucjisx0213	shift_jisx0213	iso2022jp3	tds565
riscoslatin1			

Connecting Cognos to TDV Using ODBC

Cognos is a third-party tool. These instructions are included only as a guideline; your system and the steps necessary to configure it might vary from the test system that was used in this sample. You will need to refer to Cognos documentation and perform thorough testing of your system after completing the install and configuration.

If you are using Cognos Dynamic Query Mode (DQM), you need to set up the JDBC driver to manage the connection between TDV and Cognos. Refer to you Cognos documentation for instructions on connecting to TDV.

To connect to TDV Server through Cognos on UNIX

1. Install the TDV ODBC driver on the:
   - Cognos Framework Manager Server
   - Cognos Studio clients

   The 32-bit driver is required even if running the 64-bit version of Cognos.
2. Setup ODBC drivers and define necessary environment variables. For more information, see the *TDV Administration Guide* information about using ODBC drivers with UNIX. For example, the following environment variables might need to be defined:
   - COMPOSITE_HOME
   - ODBCINI
   - ODBCINSTINI
   - LD_LIBRARY_PATH
3. Make a backup copy of existing odbc.ini and odbcinst.ini files.
4. Run the TDV driverConfig utility to generate the odbc.ini, odbcinst.ini files.
5. Change the Cognos environment to include the odbc.ini, odbcinst.ini files.
6. Add the path to the TDV ODBC driver for the following environment variable depending on your system type:

System Type	Environment Variable
Linux	LD_LIBRARY_PATH
AIX	LIBPATH

7. Configure a connection to the TDV Server on the Cognos Server.
8. Test the connectivity between TDV and Cognos.
9. Repeat the install and configuration of the TDV ODBC driver on any other machines running the Cognos Framework Manager component.
10. After publishing a view in TDV, it is not immediately available to the Cognos clients. A Cognos Framework admin needs to import the new view, create a Cognos package and publish it to the Cognos Clients.

To connect to TDV Server through Cognos on Windows

1. Install the 32-bit TDV ODBC driver on the:
   - Cognos Framework Manager Server
   - Cognos Studio clients

   The 32-bit driver is required even if running the 64-bit version of Cognos. The 32 bit ODBC manager is typically located in the C:\Windows\sysWOW64 folder and named odbcad32.exe.
2. Configure a connection to the TDV Server on the Cognos Server.
3. Test the connectivity between TDV and Cognos.
4. Repeat the install and configuration of the TDV ODBC driver on any other machines running the Cognos Framework Manager component.
5. After publishing a view in TDV, it is not immediately available to the Cognos clients. A Cognos Framework admin needs to import the new view, create a Cognos package and publish it to the Cognos Clients.

Connecting Oracle Database Gateway to TDV Using ODBC

Oracle and the Oracle heterogeneous services are a third-party tools. These instructions are included only as a guideline; your system and the steps necessary to configure it might vary from the test system that was used in this sample. You will need to refer to Oracle documentation and perform thorough testing of your system after completing the install and configuration.

To connect to TDV Server through Oracle on UNIX

1. Obtain DG4ODBC.

   DG4ODBC interacts with Oracle Heterogeneous Services to provide transparent connectivity between Oracle and non-Oracle systems. DG4ODBC is shipped with Oracle 11g. You can also download DG4ODBC from the Oracle Technology (OTN) Software Downloads Page.
2. Setup ODBC drivers and define necessary environment variables. For more information, see the *TDV Administration Guide* information about using

ODBC drivers with UNIX. For example, the following environment variables might need to be defined:

— COMPOSITE_HOME

— ODBCINI

— ODBCINSTINI

— LD_LIBRARY_PATH

3. Make a backup copy of existing odbc.ini and odbcinst.ini files.
4. Run the TDV driverConfig utility to generate the odbc.ini, odbcinst.ini files.
5. Change the Oracle environment to include the odbc.ini, odbcinst.ini files.
6. Add the path to the TDV ODBC driver for the following environment variable depending on your system type:

System Type	Environment Variable
Linux	LD_LIBRARY_PATH
AIX	LIBPATH

7. Configure a connection to the TDV Server on the Oracle Server.
8. Test the connectivity between TDV and Oracle.

Connecting MicroStrategy to TDV Using ODBC

Connecting to TDV Server through MicroStrategy varies depending on the development tool used to develop your client application. Typically, all you need to do is install and use ODBC to establish the connection.

These instructions are included as a guideline; your system and the necessary steps might vary. Refer to your development tool documentation and perform thorough testing of your system after establishing the connection between TDV Server and MicroStrategy.

MicroStrategy issues DDL statements and connects to TDV through the TDV ODBC driver. TDV creates temp tables using DDL statements in the container path that was specified.

TDV frequently adds new data sources. For updates to the supported list of data sources, see the *TDV Installation and Upgrade Guide*.

For more information for how to configure the TDV DDL feature for MicroStrategy, see “Preparing a Data Source for DDL CREATE/DROP through TDV” in the *TDV User Guide*.

You can also use ADO.Net and JDBC to connect TDV to MicroStrategy.

To connect TDV to MicroStrategy using an ODBC driver

1. Configure and publish TDV resources to the TDV DDL feature. Temporary tables are manipulated in the container path specified on the TDV DDL tab when publishing resources in TDV. For more information, see the “Publishing Resources” section of the TDV User Guide.
2. Install the ODBC driver on the client machines that have MicroStrategy and want to connect to the TDV Server. For example, install the ODBC driver using the <TDV_version>_odbc_<platform>.tar file. Or, contact MicroStrategy Technical Support to obtain the Composite<version>.PDS file.
3. Stop the MicroStrategy Intelligence Server, including all nodes of MicroStrategy Intelligence Server cluster, and disconnect all connections to the metadata from other MicroStrategy sources (such as MicroStrategy Desktop and MicroStrategy Object Manager).
4. Launch the MicroStrategy Desktop and login. Go to Database Instance Manager and edit the warehouse database instance or create a new warehouse database instance.
5. Click Upgrade.
6. Specify the driver file in the DB types script file field.
7. Click Load.
8. Move the TDV Server <version> object from the list of available databases to the list of existing databases.
9. Click OK.
10. Select the TDV Server <version> from the list of existing databases.
11. Navigate to Configuration Managers > Database Instance Manager > Database Instance > Database Connection > Advanced.
12. (Optionally) Set the Character set encoding for UNIX drivers to Non UTF-8 to fetch characters correctly for UTF-16 drivers.
13. Click OK and save the Database Connection.
14. Click OK and save the Database Instance.
15. Set up the ODBC drivers and define the necessary environment variables. For more information, see the *TDV Administration Guide* information about

using ODBC drivers with UNIX. For example, the following environment variables might need to be defined:

— COMPOSITE_HOME

— COMPOSITE_PATH

— ODBCINI

— ODBCINSTINI

— LD_LIBRARY_PATH

16. Other possible steps might include:

    — Make a backup copy of existing odbc.ini and odbcinst.ini files.

    — Run the TDV driverConfig utility to generate the odbc.ini, odbcinst.ini files.

    — Change the MicroStrategy environment to include the odbc.ini, odbcinst.ini files.

    — Add the path to the TDV ODBC driver for the following environment variable depending on your system type:

System Type	Environment Variable
Linux	LD_LIBRARY_PATH
AIX	LIBPATH

17. (Optionally) For each MicroStrategy Intelligence Server, update the MicroStrategy DTMAPPING.PDS file with the following:

    — database name

    — MSI type

18. (Optionally) Update the ODBC.sh file.

19. Define ODBC connections using DSN or Connection String methods between TDV and MicroStrategy.

20. Reload the project so that the new settings take effect. You might need to:

    — Restart the MicroStrategy Intelligence Server if using 3-tier or 4-tier modes.

    — Disconnect and re-connect the project source if using 2-tier mode.

Connecting Tableau to TDV Using ODBC

Connecting to TDV Server through Tableau varies depending on the development tool used to develop your client application. Typically, all you need to do is install and use ODBC to establish the connection.

These instructions are included as a guideline; your system and the necessary steps might vary. Refer to your development tool documentation and perform thorough testing of your system after establishing the connection between TDV Server and Tableau.

Tableau connects to TDV through the TDV ODBC driver.

Limitations

For the greatest amount of flexibility, TDV supports the standard SQL functions. Because Tableau supports several custom non-standard SQL functions, you might not be able to run certain functions against your TDV data that is displayed in Tableau.

TDV supports using CAST(TIME as TIMESTAMP). The date 1900-01-01 will be added to the TIME so that the TIME value can qualify as a TIMSTAMP data type. For example, if your time value is 12:05, your converted TIMSTAMP will be 1900-01-01 12:05.

For example, you can design a Composite view to use CAST functions to get the following conversion functions to provide the results you want in Tableau:

• DATE	• DATETIME	• FLOAT
• INT	• STR	•

To connect TDV to Tableau using an ODBC 32-bit driver

1. Install the 32-bit TDV ODBC driver on the same system that Tableau machine.

   The 32 bit ODBC manager is typically located in the %WINDIR%\sysWOW64 folder and named odbcad32.exe. For more information, see Installing the ODBC Driver, page 63.

   Tableau allows for the customization of the ODBC connection. Those instructions are at: http://kb.tableau.com/articles/knowledgebase/customizing-odbc-connections. There are two TDC files that install with TDV, their default location is <TDV_install_dir>\docs\tableau.

2. Make sure you have a DSN defined for TDV. For more information, see Adding a New System DSN, page 71.
3. Launch Tableau and login.
4. Click Data > Connect to Data.
5. Click Other Databases (ODBC).
6. Select or type the DSN that was defined for TDV. For example, select for_tableau_system.
7. Click Connect.

   Resources and data published through TDV should be viewable through your Tableau client.

   For example, using the Tableau client, you can select a TDV published table and click View data to see the data displayed in Tableau.

Connecting PowerBI to TDV Using ODBC

Connecting to TDV Server through PowerBI varies depending on the development tool used to develop your client application. Typically, all you need to do is install and use ODBC to establish the connection.

These instructions are included as a guideline; your system and the necessary steps might vary. Refer to your development tool documentation and perform thorough testing of your system after establishing the connection between TDV Server and PowerBI.

PowerBI connects to TDV through the TDV ODBC driver.

For example, you can design a Composite view to use CAST functions to get the following conversion functions to provide the results you want in Tableau:

• DATE	• DATETIME	• FLOAT
• INT	• STR	•

To connect PowerBI to TDV using an ODBC driver

1. Install the TDV ODBC driver on the same system as the PowerBI machine.
2. Make sure you have a DSN defined for TDV. For more information, see Adding a New System DSN, page 71.
3. Launch PowerBI and login.

4. Click Get Data > Type “ODBC” in filter box.
5. Select ODBC.
6. From the ODBC DSN drop-down list, select the DSN created in Step 2.
7. Provide credentials to log in to TDV.

   Resources and data published through TDV should be viewable through your PowerBI client.

   For example, using the PowerBI client, you can select a TDV published table and click on Load to view data.

Examples Using ODBC to Connect to TDV Server

This section contains examples of client applications written to access data through the TDV Server.

- PERL Code Sample for Connecting to TDV Server, page 94
- C++ Example using the Connection String (DSN-less connection), page 95
- C++ UNIX Code Sample for Connecting to TDV Server, page 98
- VBA Code Sample for Connecting to TDV Server, page 101

PERL Code Sample for Connecting to TDV Server

The following is a sample PERL script for connecting a PERL client to the TDV Server. The DSN must be configured on each client using the driverConfig utility to set the values in the odbc.ini.

```
#!/usr/bin/perl

use DBI;
use DBD::ODBC;

my $dsn="dbi:ODBC:DSN=test;";
my $dbc=DBI->connect($dsn,'admin','admin');

my $query = "select * from all_domains";
my $query_handle = $dbc->prepare($query);

$query_handle->execute();
```

```
$query_handle->bind_columns(undef, \$domain_id,
\$domain_type_name, \$domain_name, \$domain_desc);

while ($query_handle->fetch()) {
  print "$domain_id, $domain_type_name, $domain_name,
$domain_desc\n";
```

C++ Example using the Connection String (DSN-less connection)

The following example shows a small test program written in C++ for Windows that uses this connection string method. The string "dsn" uses the connection string format rather than DSN format. It is used in SQLDriverConnect call to connect to the database.

```
//

#include "stdafx.h"
int SQLSuccess(SQLRETURN rc) {
   return (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO);
}

void extract_error(const char* sqlFunc, SQLHANDLE handle,
SQLSMALLINT type) {
    SQLINTEGER i = 0;
    SQLINTEGER native;
    SQLCHAR state[7];
    SQLCHAR text[256];
    SQLSMALLINT len;
    SQLRETURN ret;

    memset(state, 0, sizeof(state));
    memset(text, 0, sizeof(text));

    do {
       ret = SQLGetDiagRec(type, handle, ++i, state, &native, text,
sizeof(text), &len);
        if (SQL_SUCCEEDED(ret)) {
            printf("%s:%ld:%ld:%s\n", state, i, native, text);
        }
    } while (ret == SQL_SUCCESS);
exit(0);
}

int fetchResultSet(SQLHSTMT stmt)
{
    SQLLEN indicator;
    SQLRETURN ret;
char buf[512];
    SQLSMALLINT columns;
    int i=0,rows=0;

    SQLNumResultCols(stmt, &columns);
    //Fetch all data
    while(1){
        ret = SQLFetch(stmt);
        if ( SQL_NO_DATA == ret ){
            break;
        }else if (!SQLSuccess(ret)) {
            extract_error("SQLFetch", stmt, SQL_HANDLE_STMT);
        }
        rows++;
        for(i=1;i<columns;i++){
           ret = SQLGetData(stmt, i, SQL_C_CHAR, buf, sizeof(buf),
&indicator);
            if (!SQLSuccess(ret)) {
                extract_error("SQLGetData", stmt, SQL_HANDLE_STMT);
            }
        }
    }
return rows;
}

int _tmain(int argc, _TCHAR* argv[])
{
printf("sizeof(SQLULEN)=%d\n",sizeof(SQLULEN));
    printf("sizeof(SQLUINTEGER)=%d\n",sizeof(SQLUINTEGER));
    printf("sizeof(SQLUSMALLINT)=%d\n",sizeof(SQLUSMALLINT));
SQLRETURN ret;
    SQLCHAR tableCat[64];
    SQLCHAR tableSchem[64];
    SQLCHAR tableName[64];
    SQLCHAR tableType[64];
    SQLCHAR remarks[64];
    SQLLEN  Str_Len;
    SQLSMALLINT colCount=0;
SQLCHAR dsn[] = "Driver={TDV
8.0};Server=localhost;Port=9401;Domain=composite;dataSource=system
;user=admin;password=admin;validateRemoteHostname=false;connectTim
eout=3000;enableFastExec=false";
//SQLCHAR query[] ="select pa11.YEAR_ID  YEAR_ID,a12.region_id
region_id,pa11.call_ctr_id  call_ctr_id,pa11.WJXBFS1  WJXBFS1 from
ZZMD00 pa11 join LU_CALL_CTR a12 on (pa11.call_ctr_id =
a12.call_ctr_id)";
    SQLCHAR query[] ="select * from ALL_TABLES";
    SQLHENV env;
    SQLHDBC dbc;
SQLHSTMT stmt;
    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
    SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (void *)
SQL_OV_ODBC3, 0);
    SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);
    ret = SQLDriverConnect(dbc, NULL, (SQLCHAR*) dsn, SQL_NTS,
NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
    if (!SQLSuccess(ret)) {
        extract_error("SQLDriverConnect", dbc, SQL_HANDLE_DBC);
}

SQLCHAR ver[512];
SQLGetInfo(dbc,
           SQL_DRIVER_NAME,
           ver,
           512,
           NULL);
printf("%s\n",ver);
SQLGetInfo(dbc,
           SQL_DRIVER_VER,
           ver,
           512,
           NULL);
printf("%s\n",ver);
SQLGetInfo(dbc,
           SQL_DBMS_NAME,
           ver,
           512,
           NULL);
printf("%s\n",ver);
SQLGetInfo(dbc,
           SQL_DBMS_VER,
           ver,
           512,
           NULL);
printf("%s\n",ver);

ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    if (!SQLSuccess(ret)) {
        extract_error("SQLAllocHandle", dbc, SQL_HANDLE_DBC);
    }
    ret =
SQLTables(stmt,(SQLCHAR*)"",0,(SQLCHAR*)"",0,(SQLCHAR*)"",0,(SQLCH
AR*)"TABLE",SQL_NTS);
    ret= SQLNumResultCols(stmt,&colCount);
    ret = SQLBindCol(stmt, 1, SQL_C_CHAR, tableCat,
sizeof(tableName), &Str_Len);
    ret = SQLBindCol(stmt, 2, SQL_C_CHAR, tableSchem,
sizeof(tableName), &Str_Len);
    ret = SQLBindCol(stmt, 3, SQL_C_CHAR, tableName,
sizeof(tableName), &Str_Len);
```

```
    ret = SQLBindCol(stmt, 4, SQL_C_CHAR, tableType,
sizeof(tableName), &Str_Len);
    ret = SQLBindCol(stmt, 5, SQL_C_CHAR, remarks,
sizeof(tableName), &Str_Len);
    while( (ret=SQLFetch(stmt))==SQL_SUCCESS){
        printf("%s\n",tableName);
    }

    ret = SQLExecDirect(stmt, (SQLCHAR*) query, SQL_NTS);
    //ret= SQLPrepareW(stmt,(SQLWCHAR*)query,10);
    ret= SQLNumResultCols(stmt,&colCount);
    if (!SQLSuccess(ret)) {
        extract_error("SQLExecDirect", stmt, SQL_HANDLE_STMT);
    }
int totalRows = fetchResultSet(stmt);
SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    SQLDisconnect(dbc);
    ret = SQLFreeHandle(SQL_HANDLE_DBC, dbc);
    ret = SQLFreeHandle(SQL_HANDLE_ENV, env);
printf("Execute query completed, total rows %d.\n",totalRows);
    fgetc(stdin);
return 0;
}
```

C++ UNIX Code Sample for Connecting to TDV Server

The following example shows a small test program written in C++ for UNIX. The string “dsn” uses the connection string format rather than DSN format. It is used in SQLDriverConnect call to connect to the database.

```
{code}
#include <sql.h>
#include <sqlext.h>
#include <stdio.h>
#include <string.h>

int SQLSuccess(SQLRETURN rc) {
return (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO);
}
void extract_error(const char* sqlFunc, SQLHANDLE handle,
SQLSMALLINT type) {
SQLINTEGER i = 0;
SQLINTEGER native;
SQLCHAR state[7];
SQLCHAR text[256];
SQLSMALLINT len;
SQLRETURN ret;
memset(state, 0, sizeof(state));
memset(text, 0, sizeof(text));
```

```
do {
ret = SQLGetDiagRec(type, handle, ++i, state, &native, text,
sizeof(text), &len);
if (SQL_SUCCEEDED(ret)) {
printf("%s:%ld:%ld:%s\n", state, i, native, text);
}
} while (ret == SQL_SUCCESS);
exit(0);
}
int fetchResultSet(SQLHSTMT stmt)
{
SQLLEN indicator;
SQLRETURN ret;
char buf[512];
SQLSMALLINT columns;
int i=0,rows=0;
SQLNumResultCols(stmt, &columns);
//Fetch all data
while(1){
ret = SQLFetch(stmt);
if ( SQL_NO_DATA == ret ){
break;
}else if (!SQLSuccess(ret)) {
extract_error("SQLFetch", stmt, SQL_HANDLE_STMT);
}
rows++;
for(i=1;i<columns;i++){
ret = SQLGetData(stmt, i, SQL_C_CHAR, buf, sizeof(buf),
&indicator);
if (!SQLSuccess(ret)) {
extract_error("SQLGetData", stmt, SQL_HANDLE_STMT);
}
}
}
return rows;
}
int main(int argc, char * argv[])
{
printf("sizeof(SQLULEN)=%d\n",sizeof(SQLULEN));
printf("sizeof(SQLUINTEGER)=%d\n",sizeof(SQLUINTEGER));
printf("sizeof(SQLUSMALLINT)=%d\n",sizeof(SQLUSMALLINT));
SQLRETURN ret;
SQLCHAR tableCat[64];
SQLCHAR tableSchem[64];
SQLCHAR tableName[64];
SQLCHAR tableType[64];
SQLCHAR remarks[64];
SQLLEN Str_Len;
SQLSMALLINT colCount=0;
```

```
SQLCHAR dsn[] = "Driver={TDV
8.0};Server=localhost;Port=9401;Domain=composite;dataSource=system
;user=admin;password=admin;validateRemoteHostname=false;connectTim
eout=3000;enableFastExec=false";
SQLCHAR query[] ="select * from ALL_TABLES";
SQLHENV env;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (void *) SQL_OV_ODBC3,
0);
SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);
ret = SQLDriverConnect(dbc, NULL, (SQLCHAR*) dsn, SQL_NTS, NULL, 0,
NULL, SQL_DRIVER_NOPROMPT);
if (!SQLSuccess(ret)) {
extract_error("SQLDriverConnect", dbc, SQL_HANDLE_DBC);
}
SQLCHAR ver[512];
SQLGetInfo(dbc,
SQL_DRIVER_NAME,
ver,
512,
NULL);
printf("%s\n",ver);
SQLGetInfo(dbc,
SQL_DRIVER_VER,
ver,
512,
NULL);
printf("%s\n",ver);
SQLGetInfo(dbc,
SQL_DBMS_NAME,
ver,
512,
NULL);
printf("%s\n",ver);
SQLGetInfo(dbc,
SQL_DBMS_VER,
ver,
512,
NULL);
printf("%s\n",ver);
ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
if (!SQLSuccess(ret)) {
extract_error("SQLAllocHandle", dbc, SQL_HANDLE_DBC);
}
ret =
SQLTables(stmt,(SQLCHAR*)"",0,(SQLCHAR*)"",0,(SQLCHAR*)"",0,(SQLCH
AR*)"TABLE",SQL_NTS);
```

```
ret= SQLNumResultCols(stmt,&colCount);
ret = SQLBindCol(stmt, 1, SQL_C_CHAR, tableCat, sizeof(tableName),
&Str_Len);
ret = SQLBindCol(stmt, 2, SQL_C_CHAR, tableSchem,
sizeof(tableName), &Str_Len);
ret = SQLBindCol(stmt, 3, SQL_C_CHAR, tableName, sizeof(tableName),
&Str_Len);
ret = SQLBindCol(stmt, 4, SQL_C_CHAR, tableType, sizeof(tableName),
&Str_Len);
ret = SQLBindCol(stmt, 5, SQL_C_CHAR, remarks, sizeof(tableName),
&Str_Len);
while( (ret=SQLFetch(stmt))==SQL_SUCCESS){
printf("%s\n",tableName);
}
ret = SQLExecDirect(stmt, (SQLCHAR*) query, SQL_NTS);
//ret= SQLPrepareW(stmt,(SQLWCHAR*)query,10);
ret= SQLNumResultCols(stmt,&colCount);
if (!SQLSuccess(ret)) {
extract_error("SQLExecDirect", stmt, SQL_HANDLE_STMT);
}
int totalRows = fetchResultSet(stmt);
SQLFreeHandle(SQL_HANDLE_STMT, stmt);
SQLDisconnect(dbc);
ret = SQLFreeHandle(SQL_HANDLE_DBC, dbc);
ret = SQLFreeHandle(SQL_HANDLE_ENV, env);
printf("Execute query completed, total rows %d.\n",totalRows);
return 0;
}
{code}
```

VBA Code Sample for Connecting to TDV Server

The following is a sample Visual Basic for Applications (VBA) Script for connecting to TDV Server from a Microsoft client (such as Excel) through ADO.

```
Sub demo()
    On Error Resume Next
    Err.Clear
    dsn = "DS-Composite"
    Set conn = CreateObject("ADODB.Connection")
    conn.Open
Driver={TDV
<ver>};host=localhost;port=9451;sso=(Disabled);uid=admin;pwd=admin
;datasource=examples;domain=composite;
If Err.Number <> 0 Then
'process error
        Exit Sub
    End If

    Err.Clear
    Set rs = CreateObject("ADODB.Recordset")
    rs.Open "SELECT * FROM CUSTOMER", conn
    If Err.Number <> 0 Then
        ' process error
        Exit Sub
    End If

    '   get column names
    For Each Column In rs.fields
        colname = Column.Name
    Next
    '   get first 100 rows
    Count = 0
    maxcount = 100

    Err.Clear
    Do While Not rs.EOF And Err.Number = 0 And Count < maxcount
        Count = Count + 1
        For Each Record In rs.fields
            colvalue = Record.Value
        Next
        rs.movenext
    Loop
End Sub
```

TIBCO Power BI Data Connector for TDV

Overview

The Power BI Connector for TIBCO(R) Data Virtualization offers self-service integration with Microsoft Power BI. The connector facilitates live access to TDV data in Power BI from the Get Data window. The connector also provides direct querying to visualize and analyze TDV data.

Getting Started

The Power BI Connector for TIBCO(R) Data Virtualization is built on top of an ODBC driver. This section discusses how to install the connector, create a DSN, and connect from Power BI.

Installing the Connector

The Power BI Connector for TIBCO(R) Data Virtualization includes comprehensive high-performance data access, real-time integration, extensive metadata discovery, and robust SQL-92 support.

Install the Connector

Complete the following steps to install the connector:

1. Download and run the Tibco Data Virtualization setup. Refer to the Installation Guide for instructions on installing TDV.
2. Setup prompts you to install Microsoft Visual C++ Redistributable for Visual Studio 2017, if it is not already installed. Proceed with the installation.
3. Once the installation is complete, click Finish.

The installation process creates a data source name (DSN) called Power BI TDV. A DSN is the name that applications use to request a connection to a data source.

Note: The Microsoft C++ Redistributable package only needs to be installed once. Subsequent installs of newer Power BI drivers will identify that the C++ Redistributable package is already installed on the machine and will not launch the installer for it again.

Creating the Data Source Name

This section describes how to edit the DSN configuration and then authenticate and connect to TDV APIs.

Editing the DSN Configuration

You can use the Microsoft ODBC Data Source Administrator to edit the DSN configuration. Note that the DSN is created during the installation process, as described in Installing the Connector.

Complete the following steps to edit the DSN configuration:

1. Select Start > Search, and enter ODBC Data Sources in the Search box.
2. Choose the version of the ODBC Administrator that corresponds to the bitness of your Power BI Desktop installation (32-bit or 64-bit).
3. Click the System DSN tab.
4. Select the system data source and click Configure.
5. Edit the information on the Connection tab and click OK.

Set the Host, Domain, User, Password, and DataSource connection properties to connect to the TDV Server.

Getting Data

After Installing the Connector and Creating the Data Source Name, you can connect to the data that you want to work with. After connecting to the data, you can edit or load the data to start building reports.

Connect to the Data

Complete the following steps to connect to the data:

1. Click Get Data.
2. Accept the warning to connect to a third-party service.
3. Select All > TIBCO(R) Data Virtualization in the Data Source Name menu.
4. Select Connect.
5. Enter the Data Source Name, Advanced Connection Properties (optional), and Advanced Options (optional).

    Note that the default Data Source Name is Power BI TDV, but you can change this name by editing the DSN configuration. See Creating the Data Source Name for more information.

6. Select a data connectivity mode and click OK. See Querying Data for more information on each mode.

   Select Import to import a copy of the data into your project. You can refresh this data on demand.

   Select DirectQuery to work with the remote data.

7. In the Navigator window, expand the Power BI TDV folder, then expand the associated schema folder to see a list of available data (tables, stored procedures, or views).

8. Select the box next to the data that you want to work with.

9. Select Load or Edit. See the next section for more information on these options.

Load or Edit the Data

After connecting to the data, load or edit the data, as described below.

- When you click Load, the connector executes the underlying query to TDV.
- When you click Edit, the Query Editor launches and a representative view of the table is presented. You can use the Query Editor to adjust the query and query results before you load the data. Right-click a column header to perform actions like the following:
  - Rename columns or tables
  - Change text to numbers
  - Remove rows
  - Set the first row as headers

Advanced Settings

The following sections detail connector settings that may be needed in advanced integrations.

Using the Power BI Connector in Multi-User Environments

Power BI loads connectors via PQX files from the [Documents]\Power BI Desktop\Custom Connectors directory (e.g. C:\Users\[User]\Documents\Power BI Desktop\Custom Connectors).

During the setup process, the PQX file will be installed to this directory for the current user. In multi-user environments, the PQX file will need to be copied to the [Documents]\Power BI Desktop\Custom Connectors folder for any user who will be using the connector.

Using the Connector

The Power BI Connector for TIBCO(R) Data Virtualization hooks into Power BI's two modes for Querying Data:

- **DirectQuery**: Visualizing data in real time by connecting directly to the data at the source.
- **Import**: Embedding data in a report, which can be refreshed on demand. This is the most common way to get data. Importing data takes advantage of the Power BI query engine.

You select the data connectivity mode when Getting Data. The following sections describe the basics of Visualizing Data, which are defining filters, aggregating data, and joining tables when working with remote data.

Querying Data

Select a data connectivity mode when you create the connection to TDV in the Get Data window. The connector fully integrates TDV connectivity into the two data connectivity modes in Power BI: DirectQuery and data import.

Using DirectQuery

Use DirectQuery mode to work with the remote data in real time, rather than a local copy. As you define filters, aggregate fields, or join tables, the connector executes the underlying queries to TDV.

Note: DirectQuery mode is limited by the DirectQueryLimit connection property.

Using Data Import

Use data import mode to save a copy of the data in your report. As you make changes to your report, Power BI executes the underlying queries to the local cache, independent of the connector.

To synchronize your report with any changes in the remote data, click Refresh from the Home menu on the ribbon.

Advanced Connection Properties (optional)

This field allows you to specify properties for the connection. For example, PropertyA=Value1;PropertyB=Value2;

Advanced Options (optional)

This field allows you to provide a SQL statement that specifies what data to return. To configure this option, expand the Advanced Options area and then, in the SQL statement field, type or paste the SQL statement. Note that SQL statements are not supported in DirectQuery mode.

You can use the following types of SQL statements:

- SELECT Statements extract data from a database. For example:

  SELECT * FROM Account

- EXECUTE Statements call procedures that are stored in a database. For example:

  EXECUTE my_proc @second = 2, @first = 1, @third = 3;

Visualizing Data

After Getting Data, you can create data visualizations in the Report view by dragging fields from the Fields pane onto the canvas. This section describes how to use visualizations to display insights that have been discovered in the data.

Creating and Working with Data Visualizations

The following example shows how to create and work with data visualizations, using a pie chart as an example.

1. Select a pie chart icon in the Visualizations pane.
2. Select a dimension in the Fields pane.
3. Select a measure in the Fields pane.

You can change sort options by clicking the ellipsis (...) button for the chart. Options to select the sort column and change the sort order are displayed.

Highlighting and Filtering Data

Highlighting and filtering change the focus on the data. Filtering removes unfocused data from visualizations; highlighting does not remove data, but instead highlights a subset of the visible data; the unhighlighted data remains visible but dimmed.

Highlight fields by clicking them. You can apply filters at the page level or at the report level. To create a filter, drag fields onto the Filters pane. Select the filter type and filter options in the Filters pane.

Creating Real-Time Visualizations

If you selected DirectQuery data connectivity mode when you created the connection, the connector builds a new SELECT WHERE clause as you change the filter.

Connection String Options

The connection string properties describe the various options that can be used to establish a connection.

Remarks

The connection string can be set to a series of "option=value" pairs, separated by semicolons. If a connection string property value has special characters such as semicolons, single quotes, spaces, etc., then you must quote the value using either single or double quotes.

Connection options are case insensitive.

To specify a location to the database where the tables, views, and stored procedures are located, set the Location property. In addition, you must also set User and Password. Caching Data can be enabled by using the appropriate options.

Connection String Options

The following is the full list of the options you can configure in the connection string for this provider.

Case Sensitive	Specifies case sensitivity in the request values.
Catalog	The name of the catalog to use.
Commit Failure	Specifies the behavior if a commit fails.

Commit Interrupt	Specifies the behavior if a commit is interrupted.
Compensate	If enabled, compensation blocks will be run if the transaction rolls back. Possible values: disabled or enabled. Default value is disabled.
Connect Timeout	The time-out for initial connection, in seconds.
Data Source	The name of the TDV data source.
Default Catalog	The default catalog for a specified connection.
Default Schema	The default schema for a specified connection.
Direct Query Limit	Limits the number of rows when using the DirectQuery mode. This helps avoid performance issues at design time.
Domain	The TDV domain to which the DataSource belongs.
Enable Failover	Specifies whether to enable failover in the case a connection fails.
Enable Fast Exec	Specifies whether to enable fast execution of queries.
Enable Reconnect On Error	Specifies cluster reconnection behavior.
Encrypt	Specifies whether to encrypt the connection using SSL.
Fetch Bytes	The maximum number of rows to fetch for a batch based on batch size, in bytes.
Fetch Rows	Maximum number of rows to fetch for a batch.
Host	The name of the server running TDV Server.
Ignore Trailing Spaces	Specifies whether to ignore trailing spaces at the end of values.
Kerberos KDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
Kerberos Realm	The Kerberos Realm used to authenticate the user with.
Kerberos SPN	The Service Principal Name for the Kerberos Domain Controller.
Locale	Value that defines the user's language and country.

Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.
Logfile	A path to the log file.
Maximum Column Size	The maximum column size.
Max Log File Size	A string specifying the maximum size in bytes for a log file (ex: 10MB). When the limit is hit, a new log is created in the same folder with the date and time appended to the end.
Max Rows	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
No Metadata	Blocks return of result-set metadata during query execution.
Optimization Prepare	Specifies whether to optimize prepare requests sent to TDV.
Other	Hidden properties needed only in specific use cases. **Note**: Streaming performance for the return of result sets can be improved by tuning the following hidden properties. These are set to true by default to improve performance. • enableFlood • stripDuplicates
Param Mode	Controls the behavior of OUT parameters for stored procedures.
Password	The user's password.
Port	The port of the TDV server.
Query Passthrough	Whether or not the provider will pass the query to TDV as-is.
Readonly	You can use this property to enforce read-only access to TDV from the provider.
Register Output Cursors	Specifies how to handle output cursors.
Request Timeout	The time-out for query commands and other requests, in seconds.
Session Timeout	Session inactivity time-out, in seconds.

SSL Client Cert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).
SSL Client Cert Password	The password for the TLS/SSL client certificate.
SSL Client Cert Subject	The subject of the TLS/SSL client certificate.
SSL Client Cert Type	The type of key store containing the TLS/SSL client certificate.
SSL Server Cert	The certificate to be accepted from the server when connecting using TLS/SSL.
SSO	The single-sign-on (SSO) type to use to authenticate.
Strip Trailing Zeros	Determines whether decimal result values are to be returned with trailing zeroes removed.
Tables	Restrict the tables reported to a subset of the available tables. For example: Tables=TableA,TableB,TableC.
Trace Folder	The absolute directory to save the trace file.
Trace Level	The level of information to log.
User	The username provided for authentication with TDV Server.
User Tokens	Authentication values that can be packaged for delivery.
Verbosity	The verbosity level that determines the amount of detail included in the log file.
Views	Restrict the views reported to a subset of the available tables. For example: Views=ViewsA,ViewsB,ViewsC.

Case Sensitive

Specifies case sensitivity in the request values.

Data Type

bool

Default Value

false

Remarks

Specifies case sensitivity in the request values. By default (false), requests are not case-sensitive.

Catalog

The name of the catalog to use.

Data Type

string

Default Value

""

Remarks

This field allows you to limit the Catalog to the one explicitly specified. If not set, the connector will retrieve the available catalogs from the TDV server.

Commit Failure

Specifies the behavior if a commit fails.

Data Type

string

Default Value

""

Remarks

Specifies the behavior if a commit fails. Possible values are: rollback or bestEffort.

Commit Interrupt

Specifies the behavior if a commit is interrupted.

Data Type

string

Default Value

""

Remarks

Specifies the behavior if a commit is interrupted. Possible values are: ignore, log, fail.

Compensate

The correcting behavior.

Data Type

string

Default Value

"disabled"

Remarks

If enabled, compensation blocks will be run if the transaction rolls back. Possible values: disabled or enabled. Default value is disabled.

Connect Timeout

The time-out for initial connection, in seconds.

Data Type

int

Default Value

0

Remarks

This property was added for AWS Data Pipeline compatibility.

The time-out for initial connection, in seconds. Use 0 (zero) for infinite time-out.

Data Source

The name of the TDV data source.

Data Type

string

Default Value

""

Remarks

Data source refers to the TDV database name published in the Data Services node.

Default Catalog

The default catalog for a specified connection.

Data Type

string

Default Value

""

Remarks

The default catalog for a specified connection.

Default Schema

The default schema for a specified connection.

Data Type

string

Default Value

""

Remarks

The default schema for a specified connection.

Direct Query Limit

Limits the number of rows when using the DirectQuery mode. This helps avoid performance issues at design time.

Data Type

string

Default Value

"10000"

Remarks

Limits the number of rows returned when using the DirectQuery Mode. This limit only applies when aggregation is not being used. Queries with SUM, MIN, MAX, GROUP BY, and so on will not be limited.

Domain

The TDV domain to which the DataSource belongs.

Data Type

string

Default Value

""

Remarks

The TDV domain to which the DataSource belongs.

Typically the domain is 'composite' for installations with locally defined users.

Enable Failover

Specifies whether to enable failover in the case a connection fails.

Data Type

bool

Default Value

false

Remarks

When set to "True" and a connection to the main host fails, the connector will attempt to connect to other machines in the cluster. The additional machines in the cluster are retrieved from the main host during the initial connection.

Enable Fast Exec

Specifies whether to enable fast execution of queries.

Data Type

bool

Default Value

false

Remarks

Values are true or false (default).

Results are processed and returned immediately (instead of round trip) when a query is submitted, potentially improving performance of low latency queries.

Enable Reconnect On Error

Specifies cluster reconnection behavior.

Data Type

bool

Default Value

false

Remarks

Specifies cluster reconnection behavior.

Encrypt

Specifies whether to encrypt the connection using SSL.

Data Type

bool

Default Value

false

Remarks

When set to true, automatically passes messages to the SSL port for processing with the TDV SSL Certificate.

Fetch Bytes

The maximum number of rows to fetch for a batch based on batch size, in bytes.

Data Type

int

Default Value

131072

Remarks

The maximum number of rows to fetch for a batch based on batch size, in bytes.

Setting FetchBytes to a very large number can cause an Out Of Memory error in the server. The value set for FetchBytes affects the memory used on the client and the TDV server, so the value should be set based on the heap size configured.

Fetch Rows

Maximum number of rows to fetch for a batch.

Data Type

int

Default Value

500

Remarks

Maximum number of rows to fetch for a batch. Set to 0 (zero) to return an unlimited number of rows.

Host

The name of the server running TDV Server.

Data Type

string

Default Value

""

Remarks

This property should be set to the name or network address of the computer running TDV Server.

Ignore Trailing Spaces

Specifies whether to ignore trailing spaces at the end of values.

Data Type

bool

Default Value

false

Remarks

Specifies whether to ignore trailing spaces at the end of values.

Kerberos KDC

The Kerberos Key Distribution Center (KDC) service used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using Windows Authentication. The connector will request session tickets and temporary session keys from the Kerberos Key Distribution Center (KDC) service. The Kerberos Key Distribution Center (KDC) service is conventionally colocated with the domain controller. If Kerberos KDC is not specified the connector will attempt to detect these properties automatically from the following locations:

- **Java System Properties**: Kerberos settings can be configured in Java using the config file krb5.conf, or using the system properties java.security.krb5.realm and java.security.krb5.kdc. The connector will use the system settings if KerberosRealm and KerberosKDC are not explicitly set.
- **Domain Name and Host**: The connector will infer the Kerberos Realm and Kerberos KDC from the configured domain name and host as a last resort.

Note: Windows authentication is supported in JRE 1.6 and above only.

Kerberos Realm

The Kerberos Realm used to authenticate the user with.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using SPNEGO or Windows Authentication. The Kerberos Realm is used to authenticate the user with the Kerberos Key Distribution Service (KDC). The Kerberos Realm can be configured by an administrator to be any string, but conventionally it is based on the domain name. If Kerberos Realm is not specified the connector will attempt to detect these properties automatically from the following locations:

- **Java System Properties**: Kerberos settings can be configured in Java using a config file (krb5.conf) or using the system properties java.security.krb5.realm and java.security.krb5.kdc. The connector will use the system settings if KerberosRealm and KerberosKDC are not explicitly set.
- **Domain Name and Host**: The connector will infer the Kerberos Realm and Kerberos KDC from the user-configured domain name and host as a last resort. This might work in some Windows environments.

Note: Kerberos-based authentication is supported in JRE 1.6 and above only.

Kerberos SPN

The Service Principal Name for the Kerberos Domain Controller.

Data Type

string

Default Value

""

Remarks

If the Service Principal Name on the Kerberos Domain Controller is not the same as the URL that you are authenticating to, set the Service Principal Name here.

Locale

Value that defines the user's language and country.

Data Type

string

Default Value

""

Remarks

Value that defines the user's language and country.

Location

A path to the directory that contains the schema files defining tables, views, and stored procedures.

Data Type

string

Default Value

""

Remarks

The path to a directory which contains the schema files for the connector (.rsd files for tables and views, .rsb files for stored procedures). The Location property is only needed if you would like to customize definitions (e.g., change a column name, ignore a column, etc.) or extend the data model with new tables, views, or stored procedures.

The schema files are deployed alongside the connector assemblies. You must also ensure that Location points to the folder that contains the schema files. The folder location can be a relative path from the location of the executable.

Logfile

A path to the log file.

Data Type

string

Default Value

""

Remarks

For more control over what is written to the log file, take a look at Verbosity.

Maximum Column Size

The maximum column size.

Data Type

string

Default Value

"16000"

Remarks

Some tools restrain the largest size of a column or the total size of all the columns selected. You can set the MaximumColumnSize to overcome these schema-based restrictions. The connector will not report any column to be larger than the MaximumColumnSize.

Set a MaximumColumnSize of zero to eliminate limits on column size, as shown in the following example:

SQLSetConnectAttr(hdbc, 20002, (SQLPOINTER)2048, 0);

The following are a few examples of how you can use this property to avoid compatibility issues with several tools:

- Oracle ODBC Gateway: Set MaximumColumnSize=4000 to avoid the ORA-28562 data truncation error. Note that Oracle ODBC Gateway additionally requires that you set the MapToWVarchar connection property to false.
- Microsoft Access: Set MaximumColumnSize=255 to report string fields as TEXT instead of MEMO in Access. MEMO fields have no length limit but have restrictions on joins and filters. TEXT fields have a fixed length but support more functionality in Access tables.

Max Log File Size

A string specifying the maximum size in bytes for a log file (ex: 10MB). When the limit is hit, a new log is created in the same folder with the date and time appended to the end.

Data Type

string

Default Value

"20MB"

Remarks

A string specifying the maximum size in bytes for a log file (ex: 10MB). When the limit is hit, a new log is created in the same folder with the date and time appended to the end. The default limit is 20MB. Values lower than 100kB will use 100kB as the value instead.

Max Rows

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

Data Type

string

Default Value

"-1"

Remarks

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

No Metadata

Blocks return of result-set metadata during query execution.

Data Type

bool

Default Value

false

Remarks

Blocks return of result-set metadata during query execution.

Optimization Prepare

Specifies whether to optimize prepare requests sent to TDV.

Data Type

bool

Default Value

true

Remarks

When set to "True" (default), the connector will submit the query in a single request to TDV.

When set to "False", the connector will submit an initial prepare request to TDV.

Other

Hidden properties needed only in specific use cases.

Data Type

string

Default Value

""

Remarks

The properties listed below are available for specific use cases. Normal driver use cases and functionality should not require these properties.

Specify multiple properties in a semicolon-separated list.

Integration and Formatting

DefaultColumnSize	Sets the default length of string fields when the data source does not provide column length in the metadata. The default value is 2000.
ConvertDateTimeToGMT	Whether to convert date-time values to GMT, instead of the local time of the machine.
RecordToFile=filename	Records the underlying socket data transfer to the specified file.

Param Mode

Controls the behavior of OUT parameters for stored procedures.

Data Type

string

Default Value

"normal"

Remarks

Controls the behavior of OUT parameters for stored procedures.

Valid values are:

normal	Report OUT parameters in procedure metadata as OUT parameters.
return	Report OUT parameters as return values.
omit	Omit OUT parameters from metadata.
omitCursors	Omit output cursors from metadata.

Password

The user's password.

Data Type

string

Default Value

""

Remarks

The password provided for authentication with the TDV Server.

Port

The port of the TDV server.

Data Type

int

Default Value

9401

Remarks

The port of the server hosting the TDV Server.

Default is 9401 (plaintext) and the SSL protected port is 9403.

Query Passthrough

Whether or not the provider will pass the query to TDV as-is.

Data Type

bool

Default Value

false

Remarks

Whether or not the connector will pass the query to TDV as-is.

Readonly

You can use this property to enforce read-only access to TDV from the provider.

Data Type

bool

Default Value

false

Remarks

If this property is set to true, the connector will allow only SELECT queries. INSERT, UPDATE, DELETE, and stored procedure queries will cause an error to be thrown.

Register Output Cursors

Specifies how to handle output cursors.

Data Type

bool

Default Value

false

Remarks

Specifies how to handle output cursors.

Valid values are:

true	Bind or register output cursors as output parameters.
false	Do not bind or register output cursors as output parameters; instead, use SQLMoreResults or Statement.getMoreResults() to access the cursors.

Request Timeout

The time-out for query commands and other requests, in seconds.

Data Type

int

Default Value

0

Remarks

The time-out for query commands and other requests, in seconds.

Session Timeout

Session inactivity time-out, in seconds.

Data Type

int

Default Value

0

Remarks

Session inactivity time-out, in seconds. Set to 0 (zero) for infinite time-out.

SSL Client Cert

The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).

Data Type

string

Default Value

""

Remarks

The name of the certificate store for the client certificate.

The SSLClientCertType field specifies the type of the certificate store specified by SSLClientCert. If the store is password protected, specify the password in SSLClientCertPassword.

SSLClientCert is used in conjunction with the SSLClientCertSubject field in order to specify client certificates. If SSLClientCert has a value, and SSLClientCertSubject is set, a search for a certificate is initiated. Please refer to the SSLClientCertSubject field for details.

Designations of certificate stores are platform-dependent.

The following are designations of the most common User and Machine certificate stores in Windows:

MY	A certificate store holding personal certificates with their associated private keys.
CA	Certifying authority certificates.
ROOT	Root certificates.
SPC	Software publisher certificates.

In Java, the certificate store normally is a file containing certificates and optional private keys.

When the certificate store type is PFXFile, this property must be set to the name of the file. When the type is PFXBlob, the property must be set to the binary contents of a PFX file (i.e. PKCS12 certificate store).

SSL Client Cert Password

The password for the TLS/SSL client certificate.

Data Type

string

Default Value

""

Remarks

If the certificate store is of a type that requires a password, this property is used to specify that password in order to open the certificate store.

SSL Client Cert Subject

The subject of the TLS/SSL client certificate.

Data Type

string

Default Value

"*"

Remarks

When loading a certificate the subject is used to locate the certificate in the store.

If an exact match is not found, the store is searched for subjects containing the value of the property.

If a match is still not found, the property is set to an empty string, and no certificate is selected.

The special value "*" picks the first certificate in the certificate store.

The certificate subject is a comma separated list of distinguished name fields and values. For instance "CN=www.server.com, OU=test, C=US, E=support@cdata.com". Common fields and their meanings are displayed below.

Field	Meaning
CN	Common Name. This is commonly a host name like www.server.com.
O	Organization
OU	Organizational Unit
L	Locality
S	State
C	Country
E	Email Address

If a field value contains a comma it must be quoted.

SSL Client Cert Type

The type of key store containing the TLS/SSL client certificate.

Data Type

string

Default Value

""

Remarks

This property can take one of the following values:

USER - default	For Windows, this specifies that the certificate store is a certificate store owned by the current user. Note: This store type is not available in Java.
MACHINE	For Windows, this specifies that the certificate store is a machine store. Note: this store type is not available in Java.
PFXFILE	The certificate store is the name of a PFX (PKCS12) file containing certificates.
PFXBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in PFX (PKCS12) format.
JKSFILE	The certificate store is the name of a Java key store (JKS) file containing certificates. Note: this store type is only available in Java.
JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in Java key store (JKS) format. Note: this store type is only available in Java.
PEMKEY_FILE	The certificate store is the name of a PEM-encoded file that contains a private key and an optional certificate.
PEMKEY_BLOB	The certificate store is a string (base64-encoded) that contains a private key and an optional certificate.
PUBLIC_KEY_FILE	The certificate store is the name of a file that contains a PEM- or DER-encoded public key certificate.
PUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains a PEM- or DER-encoded public key certificate.

SSHPUBLIC_KEY_FILE	The certificate store is the name of a file that contains an SSH-style public key.
SSHPUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains an SSH-style public key.
P7BFILE	The certificate store is the name of a PKCS7 file containing certificates.
PPKFILE	The certificate store is the name of a file that contains a PPK (PuTTY Private Key).
XMLFILE	The certificate store is the name of a file that contains a certificate in XML format.
XMLBLOB	The certificate store is a string that contains a certificate in XML format.

SSL Server Cert

The certificate to be accepted from the server when connecting using TLS/SSL.

Data Type

string

Default Value

""

Remarks

If using a TLS/SSL connection, this property can be used to specify the TLS/SSL certificate to be accepted from the server. Any other certificate that is not trusted by the machine will be rejected.

This property can take the forms:

Description	Example
A full PEM Certificate (example shortened for brevity)	-----BEGIN CERTIFICATE----- MIIChTCCAe4CAQAwDQYJKoZIhv......Qw== -----END CERTIFICATE-----

Description	Example
A path to a local file containing the certificate	C:\cert.cer
The public key (example shortened for brevity)	-----BEGIN RSA PUBLIC KEY----- MIGfMA0GCSq......AQAB -----END RSA PUBLIC KEY-----
The MD5 Thumbprint (hex values can also be either space or colon separated)	ecadbdda5a1529c58a1e9e09828d70e4
The SHA1 Thumbprint (hex values can also be either space or colon separated)	34a929226ae0819f2ec14b4a3d904f801cbb150d

If not specified, any certificate trusted by the machine will be accepted. Use '*' to signify to accept all certificates (not recommended for security concerns).

Use '*' to signify to accept all certificates. Note that this is not recommended due to security concerns.

SSO

The single-sign-on (SSO) type to use to authenticate.

Data Type

string

Default Value

"Disable"

Remarks

The single-sign-on (SSO) type to use to authenticate. Valid values are: Disable, Kerberos, and NTLM.

Valid on Windows platform only.

Default is "Disable" which forces the client to provide a user and password to authenticate.

Strip Trailing Zeros

Determines whether decimal result values are to be returned with trailing zeroes removed.

Data Type

bool

Default Value

false

Remarks

Determines whether decimal result values are to be returned with trailing zeroes removed.

Tables

Restrict the tables reported to a subset of the available tables. For example: Tables=TableA,TableB,TableC.

Data Type

string

Default Value

""

Remarks

Listing the tables from some databases can be expensive. Providing a list of tables in the connection string improves the performance of the connector.

This property can also be used as an alternative to automatically listing views if you already know which ones you want to work with and there would otherwise be too many to work with.

Specify the tables you want in a comma-separated list. For example:
Tables=TableA,TableB,TableC

Trace Folder

The absolute directory to save the trace file.

Data Type

string

Default Value

""

Remarks

The absolute directory to save the trace file.

Trace Level

The level of information to log.

Data Type

string

Default Value

"error"

Remarks

The level of information to log. Valid values are: off, fatal, error (default), warn, info, debug, and all.

User

The username provided for authentication with TDV Server.

Data Type

string

Default Value

""

Remarks

The username provided for authentication with TDV Server.

User Tokens

Authentication values that can be packaged for delivery.

Data Type

string

Default Value

""

Remarks

Authentication values that can be packaged for delivery.

The URL can pass the user_tokens property to the server at the init command, in the form: " user_tokens=(" NAME "=" VALUE ("," NAME "=" VALUE)* ")"

Verbosity

The verbosity level that determines the amount of detail included in the log file.

Data Type

string

Default Value

"1"

Remarks

The verbosity level determines the amount of detail that the connector reports to the Logfile. Verbosity levels from 1 to 5 are supported. These are described below:

1	Setting Verbosity to 1 will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.
2	Setting Verbosity to 2 will log everything included in Verbosity 1 and additional information about the request, if applicable.
3	Setting Verbosity to 3 will additionally log the body of the request and the response.
4	Setting Verbosity to 4 will additionally log transport-level communication with the data source. This includes SSL negotiation.
5	Setting Verbosity to 5 will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands.

The Verbosity should not be set to greater than 1 for normal operation. Substantial amounts of data can be logged at higher verbosities, which can delay execution times.

Views

Restrict the views reported to a subset of the available tables. For example: Views=ViewsA,ViewsB,ViewsC.

Data Type

string

Default Value

""

Remarks

Listing the views from some databases can be expensive. Providing a list of views in the connection string improves the performance of the connector.

This property can also be used as an alternative to automatically listing views if you already know which ones you want to work with and there would otherwise be too many to work with.

Specify the views you want in a comma-separated list. For example: For example: Views=ViewsA,ViewsB,ViewsC

Connecting to TDV Server through Web Interfaces

This topic describes how to retrieve data through a SOAP, REST, and OData client program.

Connecting to TDV Server through SOAP

Web service clients can access TDV-defined Web services published through SOAP over HTTP. This topic describes how to view the WSDL of a TDV Web service published by SOAP and verify that the Web service is ready.

TDV requires that messages sent to the TDV Server are from identifiable source and can be authenticated.

View the SOAP of a TDV Web service from Studio

1. In Studio, select the data service under Web Services in the resource tree.
2. Right-click and select Open.
3. Select the SOAP tab.
4. Expand WSDL URLs under the Service portion of the screen.
5. Copy one or more of the URLs that is displayed.
6. Open the development tool or file where you are developing your SOAP client, and paste the URL.

View the SOAP of a TDV Web service using a URL in a browser

1. In a browser, enter the services URL of the data service. You can use Studio to locate the URL of your service.

   Or type the URL using the following format:

http://<Host>:<HTTP_Port>/services/<Folders>/<DataServiceName>.wsdl
https://<Host>:<HTTPs_Port>/services/<Folders>/<DataServiceName>.wsdl:

- — **`<Host>`** is the name of the machine where TDV Server is running. If you are on the same computer as TDV, the computer name would be **`localhost`**.
- — **`<HTTP_Port> or <HTTPS_Port>`** is the number of the port where the Web service is published.
- — <**`Folders`**> is any optional folder or set of folders containing the Data Service.
- — **`<DataServiceName>`** is the name for the **`WSDL`**-transformed data service.

You can use this URL (or any accessible WSDL URL) as a data source to demonstrate the availability of the data, or to demonstrate TDV introspection into WSDL data sources.

2. Click the ENTER key.

   The WSDL of the Data Service is displayed in the browser.

3. If the WSDL is as you want it, the Web service is now ready for importing into your client application.
4. Open the development tool or file where you are developing your SOAP client, and paste the URL.
5. If the published content from TDV does not provide binary parameters, attempting to set Force MTOM to true will not work as expected.

View the SOAP of a Legacy TDV Web service from Studio

1. In Studio, select the data service under Web Services in the resource tree.
2. Right-click and select View WSDL.
3. Type in a valid username and password with access to the WSDL.

   A browser window opens, displaying the currently published WSDL.

4. If the WSDL is as you want it, the Web service is now ready for importing into a client application.
5. Copy the URL from the URL address field at the top of your browser.
6. Open the development tool or file where you are developing your SOAP client, and paste the URL.

SOAP Message Compression

TDV Web services supports GZIP compression of the SOAP message body.

The client making an HTTP request of a published TDV Web service needs to signal that it supports GZIP compression with a request header that includes:

```
Accept-Encoding: gzip
```

The response message sent to the client has a compressed message body, and the HTTP header looks like the following:

```
Content-Type: text/xml; charset=utf-8
Content-Encoding: gzip
Transfer-Encoding: chunked
Server: Jetty(6.2.11)
```

SOAP Message Optimization

TDV Web services supports the FastInfoset encoding of SOAP messages.

The client application making an HTTP request of a published TDV Web service needs to signal that it supports FastInfoset encoding with a request header that includes:

```
Accept: application/fastinfoset
```

The response message sent to the client application has a binary message body, and the HTTP header looks like the following:

```
Content-Type: application/fastinfoset
Server: Jetty(6.2.11)
```

Connecting to TDV Server through REST

Connecting to TDV Server through REST varies depending on the development tool that you are using to develop your client application, but typically all you need to do is use the Service URL to establish the connection. There are no additional drivers to install.

These instructions are included as a guideline; your system and the necessary steps might vary. Refer to your development tool documentation and perform thorough testing of your system after establishing the connection between TDV Server and REST.

TDV requires that messages sent to the TDV Server are from identifiable source and can be authenticated.

To define the connection between TDV Server and REST

1. In Studio, select the data service under Web Services in the resource tree.
2. Right-click and select Open.
3. Select the REST tab.
4. Expand Endpoint URLs under the Operations portion of the screen.
5. Copy one or more of the URLs that is displayed.
6. Open the development tool or file where you are developing your REST client, and paste the URL.

Connecting to TDV Server through OData

The Open Data Protocol (OData) is an open Web protocol for querying and updating data.

Connecting to TDV Server through OData varies depending on the development tool that you are using to develop your client application, but typically all you need to do is use the Service URL to establish the connection. There are no additional drivers to install.

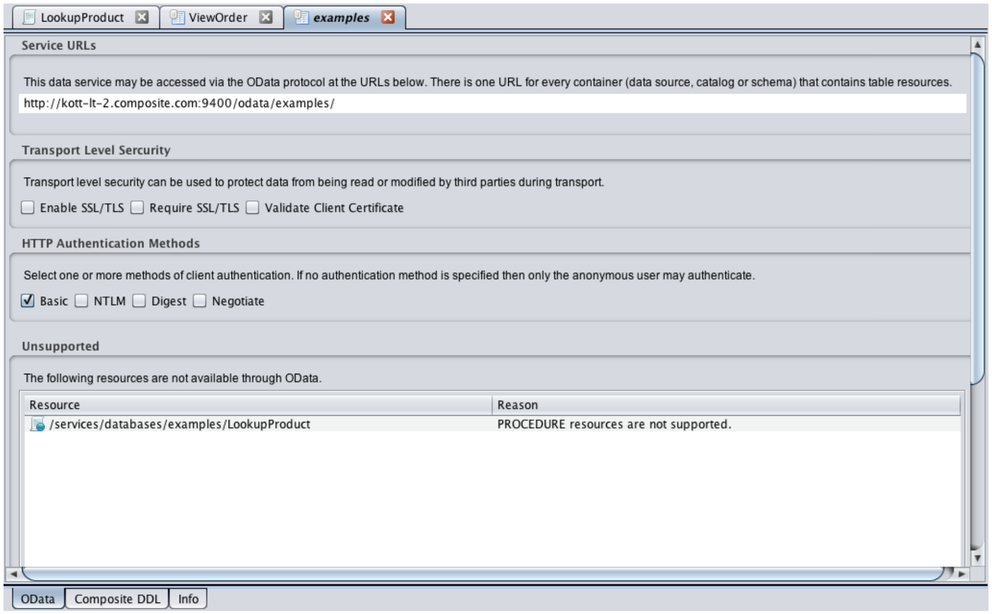
These instructions are included as a guideline; your system and the necessary steps might vary. Refer to your development tool documentation and perform thorough testing of your system after establishing the connection between TDV Server and OData.

OData with TDV:

- Allows datasource query over HTTP.
- Returns results in XML.
- Similar to ODBC over HTTP.
- Comes for free when relational resource published in virtual database.
- Works with tables and views.
- Requires that primary key metadata is defined.
- Does not work with procedures.

To define the connection between TDV Server and OData

1. Open the Data Service in Studio.
2. Navigate to the OData configuration panel.



Service URLs

This data service may be accessed via the OData protocol at the URLs below. There is one URL for every container (data source, catalog or schema) that contains table resources.

http://kott-lt-2.composite.com:9400/odata/examples/

Transport Level Sercurity

Transport level security can be used to protect data from being read or modified by third parties during transport.

☐ Enable SSL/TLS ☐ Require SSL/TLS ☐ Validate Client Certificate

HTTP Authentication Methods

Select one or more methods of client authentication. If no authentication method is specified then only the anonymous user may authenticate.

☑ Basic ☐ NTLM ☐ Digest ☐ Negotiate

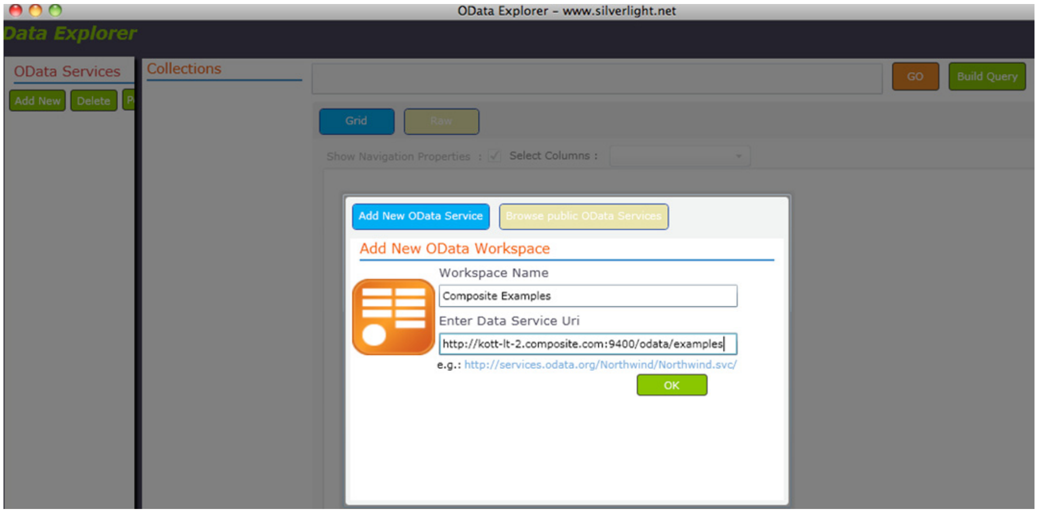
Unsupported

The following resources are not available through OData.

Resource	Reason
/services/databases/examples/LookupProduct	PROCEDURE resources are not supported.

3. Copy the URL listed in the Service URLs section.

4. Open the development tool or file where you are developing your OData client, and paste the URL. For example, when using OData Explorer:



Add New OData Workspace

Workspace Name: Composite Examples

Enter Data Service Uri: http://kott-lt-2.composite.com:9400/odata/examples

e.g.: http://services.odata.org/Northwind/Northwind.svc/

Connecting to TDV Server through ADO.NET

You can use the native ADO.NET driver functionality on Windows operating systems to develop or consume TDV resources.

You can use Microsoft Visual Studio and other third-party software to develop solutions that use resources defined by the TDV through the ADO.NET driver interface. The .NET Data provider is written in managed C# code and provides a native implementation of ADO.NET API.

These topics are included:

Setting Up the ADO.NET Driver

This section includes the following:

Client-Side ADO.NET Driver Support

The TDV ADO.NET driver can be installed, uninstalled, or re-installed. It can support 32-bit and 64-bit Windows operation systems. TDV Software supports native ADO.NET driver functionality on the following Windows operating systems.

- Windows 7 SP1 Professional
- Windows 7 SP1 Professional x64
- Windows 8.1 Professional x64
- Windows 10 v1803
- Windows Server 2012 R2
- Windows Server 2016 R2
- Windows Server 2019

TDV supports communication and use with:

- Visual Studio 2012, 2013 and 2015

Installing the ADO.NET Driver

The TDV ADO.NET driver installation package for Windows operating systems is named, TIB_tdv_ADONet_<TDV Version>_<ADO.Net version>.msi. For example, in 8.4 release of TDV, the driver is called as TIB_tdv_ADONet_8.4.0_4.5.msi. Obtain this from the client driver zip file.

To install the ADO.NET driver

1. Make sure Visual Studio and any other interface that accesses the ADO.Net driver is closed.
2. If an older version of the driver exists, it is recommended that you uninstall that version, before installing the current version.
3. Double click or open TIB_tdv_ADONet_<TDV Version>_<ADO.Net version>.msi. For example, in 8.4 release of TDV, the driver is called as TIB_tdv_ADONet_8.4.0_4.5.msi

During the last installation step, the Visual Studio command table is rebuilt to complete the process. This part of the installation process takes a while to complete.

Uninstalling and Repairing ADO.NET

You can uninstall ADO.NET or repair it.

To uninstall or repair the ADO.NET driver

1. Run the TIB_tdv_ADONet_<TDV Version>_<ADO.Net version>.msi installer on a computer that has an ADO.NET driver installed. For example, in 8.4 release of TDV, the driver is called as TIB_tdv_ADONet_8.4.0_4.5.msi
2. The installer automatically gives you the option to uninstall or repair the TDV ADO.NET driver.

You can also use the Windows Add/Remove Programs from control panel to remove the ADO.NET driver.

Updating the ADO.NET Driver

After updating your driver, you might need to update your client application code.

To update the ADO.NET driver

1. Make sure there is no application using and holding ADO.NET driver.
2. Uninstall the old ADO.NET driver.
3. Install the new ADO.NET driver.
4. Update connection settings in the client and development tools.

Configure an ADO.NET Connection to a Client Restricted Server

OEM installations of the TDV Server that have a client-restricted license might require that all client connections with TDV present a valid session token to negotiate the connection. Only clients submitting a valid session token are able to connect with client-restricted OEM TDV Servers.

The ADO.NET client uses the TDV ADO.NET driver to connect with TDV. The ADO.NET client must incorporate a valid session token into the connection string for the license restricted OEM TDV Servers.

To provide a valid session token

1. Locate the sessionToken string value, sent with the TDV Server license file, that enables licensed OEM use of the TDV Server through authorized clients.
2. Add the sessionToken into the ADO.NET client ConnectionString field so that the string is reported like the following:

```
host=10.1.2.123;port=9401;domain=composite;dataSource=TEST;session
Token=ffa29823a2a8e340f20f15048aeebced746387d235abc12;
```

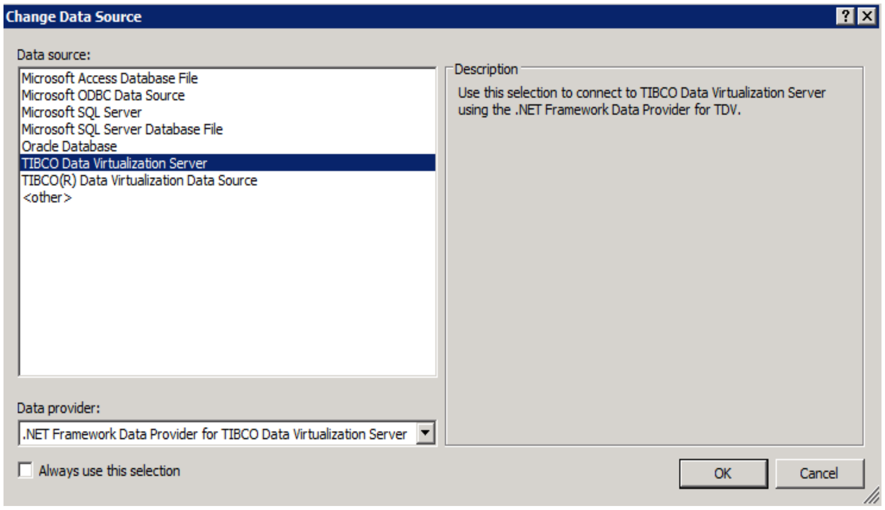
Adding and Configuring a Connection to TDV in Visual Studio

To communicate with a published TDV data service, provide the ADO.NET driver with the appropriate properties for host, port, user, password, and domain to create a connection.

To create and configure a connection

1. Open the Visual Studio Server Explorer pane by choosing the Server Explorer selection from the View menu.
2. Right-click the Data Connections node, select the option to Add Connection, and click OK.
3. Select TIBCO Data Virtualization Server as the data source.

   If TDV is not displayed, then either the driver was not installed or Visual Studio must be restarted to recognize the driver.



Data source:
- Microsoft Access Database File
- Microsoft ODBC Data Source
- Microsoft SQL Server
- Microsoft SQL Server Database File
- Oracle Database
- TIBCO Data Virtualization Server
- TIBCO(R) Data Virtualization Data Source
- <other>

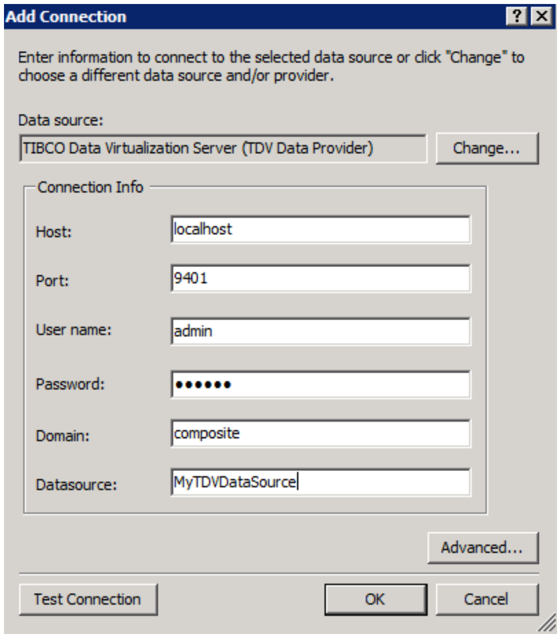
Description

Use this selection to connect to TIBCO Data Virtualization Server using the .NET Framework Data Provider for TDV.

Data provider: .NET Framework Data Provider for TIBCO Data Virtualization Server

Always use this selection

4. Click Continue and enter the connection information for the selected data source.



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: TIBCO Data Virtualization Server (TDV Data Provider)

Connection Info

Host: localhost

Port: 9401

User name: admin

Password: ••••••

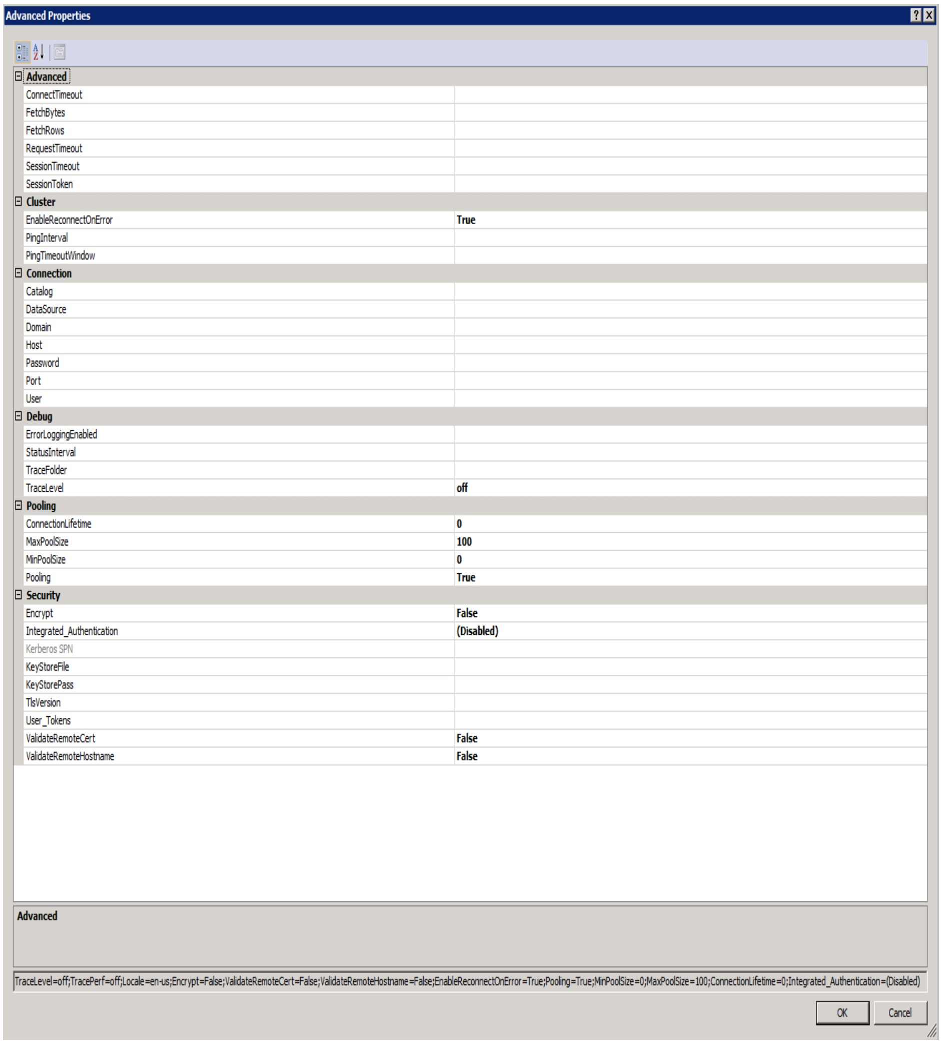
Domain: composite

Datasource: MyTDVDataSource

Field	Description
Host	TDV Server host name or IP octet. Use localhost if you plan to use Visual Studio and TDV Server on the same local computer.
Port	Use the number of the JDBC or ODBC open port (default: 9401) or the SSL protected port (default: 9403). For use of SSL, Server certificates must be installed separately.
User Name	The TDV user name.
Password	The TDV password.
Domain	The TDV domain to which this data source belongs. Typically, that domain is composite for installations with locally defined users.
Datasource	Data source refers to the TDV database name published in the Data Services node.
Catalog	Specifies the catalog that is used for the connection.

Note: The User Name, Password, and Domain fields are not used for Kerberos or NTLM authentication because the authentication status is negotiated according to the presence of a token and communication with the Kerberos authentication server.

5. Click Test Connection to make sure that you can connect to the TDV data source.
6. Click Advanced.
7. Use the Advanced Properties panel to configure the connection, debug, and security settings.



Advanced Properties

Advanced	
ConnectTimeout	
FetchBytes	
FetchRows	
RequestTimeout	
SessionTimeout	
SessionToken	
Cluster	
EnableReconnectOnError	True
PingInterval	
PingTimeoutWindow	
Connection	
Catalog	
DataSource	
Domain	
Host	
Password	
Port	
User	
Debug	
ErrorLoggingEnabled	
StatusInterval	
TraceFolder	
TraceLevel	off
Pooling	
ConnectionLifetime	0
MaxPoolSize	100
MinPoolSize	0
Pooling	True
Security	
Encrypt	False
Integrated_Authentication	(Disabled)
Kerberos SPN	
KeyStoreFile	
KeyStorePass	
TlsVersion	
User_Tokens	
ValidateRemoteCert	False
ValidateRemoteHostname	False

Advanced

```
TraceLevel=off;TracePerf=off;Locale=en-us;Encrypt=False;ValidateRemoteCert=False;ValidateRemoteHostname=False;EnableReconnectOnError=True;Pooling=True;MinPoolSize=0;MaxPoolSize=100;ConnectionLifetime=0;Integrated_Authentication=(Disabled)
```

Note: SSL security is configured using the Security Encrypt, KeyStoreFile, and KeystorePass parameters described below.

Parameter	Value description
Advanced	
AccessToken	The authorization tokens used for OAuth2 authentication. The tokens are represented in a specific format - <header>.<payload>.<signature>. Each of the parts of the token is in a JSON format. The access token is used in place of id/password credentials, with a limited lifetime & privileges.
AccessTokenType	The type of the AccessToken. JWT (JSON Web Token) is the default supported format. JWT token is a self-contained JSON form and ideal for federation.
ConnectTimeout	The number of seconds the client waits for a connection to be established or to fail. A value of 0 disables the timeout.
FetchBytes	Maximum number of rows to fetch for a batch based on batch size, in bytes. Setting fetchBytes to a very large number can cause an Out Of Memory error in the server. The value set for fetchBytes affects the memory used on the JDBC client and the TDV server, so the value should be set based on the heap size configured.
FetchRows	The maximum number of rows fetched from TDV at one time. Note: There is no relationship between FetchRows and FetchBytes. When the dbchannel gets a record, it calculates the number of fetched rows and fetched bytes. If the number of fetched rows is greater than fetchRows or the number of fetched bytes is greater than fetchBytes, dbchannel stops fetching rows and returns all fetched records.
RequestTimeout	The number of seconds the client waits for the TDV Server to return a request. A value of `0` disables the timeout.
SessionTimeout	Timeout for session inactivity on the server. This setting gives the TDV Server an indication of how long a session should be maintained if the connection with the client is lost without the server being notified.
SessionToken	The location of the session token used to authenticate to the sever.
Cluster	

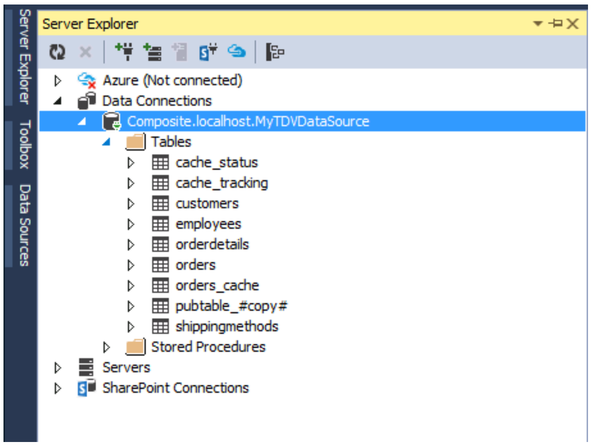
Parameter	Value description
EnableReconnectOnError	Default value (false) results in an exception if the connection object dies. If you set it to true, the driver tries to create a new connection to the same server when the connection dies. When Active Clustering is in use, set this value to true, so that any failure to connect automatically initiates another attempt to connect to the server.
PingInterval	The TDV ADO.NET driver lets you configure a ping mechanism to assess TDV status after it sends a command to the server. After the driver sends a request to the server, PingInterval initiates ping verifications repeatedly at the specified interval. Verification of status by pings helps the server avoid lengthy response times for verification of connection status—for example, when the connection might have been lost just after a command was issued. PingInterval is the number of seconds between consecutive ping status checks. The default value is 0, which disables ping verification.
PingTimeoutWindow	This should be greater than or equal to the value of PingInterval. It means the ping timeout fails. From the time that common command send to the server, the driver sends continuous ping commands to the server. If the ping still fails after PingTimeoutWindow, an exception is thrown and the connection is closed. Otherwise, the driver waits and sends ping commands to the server to check the server's status. Default value is 0, which means this option is not in use.
Connection	Properties in this section are values that were set in Adding and Configuring a Connection to TDV in Visual Studio, page 149. The values of the catalog, data source name, domain, host, port, and user are populated from the values entered on that screen, and they can be changed here.
Debug	
ErrorLoggingEnabled	Enable ADO.NET driver log.
StatusInterval	Log thread and connection status with a specified time interval in seconds.
TraceFolder	Absolute directory to save trace file, the trace file name is "CsOdbcDebug_"+<DSN Name>+".log". the default folder is C:\ or $COMPOSITE_HOME.

Parameter	Value description
TraceLevel	Valid values are off, fatal, error (this is the default), debug, warn, info, debug, and all. The valid values for client-side log settings are off, fatal, error (default), warn, info, debug, all, stdout. On UNIX-based platforms, the log file CsOdbcDebug.log is created in the directory specified by the environment variable COMPOSITE_HOME.
Pooling	
ConnectionLifeTime	Setting ConnectionLifeTime=0 means each connection will be closed as soon as it is used. When using connection pool if you want to reuse the connection by putting the connection back to the connection pool set 'connectionlifetime' value to a value greater than zero. The Unit for ConnectionLifeTime is second. For example ConnectionLifeTime=60000 means 60000 seconds
MaxPoolSize	Sets the maximum number of connections that are opened in the same pool at the same time. If the maximum is reached and no usable connection is available, subsequent requests are queued until a connection is available.
MinPoolSize	Sets the minimum number of connections that is maintained even if inactive to avoid the time cost of recreating new connections for a new request.
Pooling	When true, inactive connections are saved and reused as necessary.
Security	
Encrypt	True or False. Used for SSL security. Set to True to enable SSL security. The default is False.
Integrated_Authentication	Specified the integrate authentication method with three values: '(Disabled)', 'Kerberos', 'NTLM'.
Kerberos SPN	A service principal name (SPN) used by Kerberos authentication to associate a service instance with a service logon account. This allows for service authenticate of an account even if the client does not have the account name.
KeyStoreFile	Used for SSL security. Specifies the keystore file to use for verification. The file is in the PKCS#12 format. The default keystore could be found at apps/ADO.NET/ Security/cis_ado_keystore.pfx, which includes the client certificate and private key.

Parameter	Value description
KeyStorePass	Used for SSL security. Specifies the password for the KeyStoreFile. The default value is 'changeit'.
User_Tokens	Authentication values that can be packaged for delivery.
TlsVersion	The TLS Version information.
ValidateRemoteCert	True or False. The default is False.
ValidateRemoteHostname	True or False. The default is False.

8. Click **OK** twice to finish the configuration.

After defining the Connection profile settings, the newly created connection to the TDV data source is displayed in the Visual Studio Server Explorer. You can work with the TDV data source using the standard Server Explorer interface.



Server Explorer
- Azure (Not connected)
- Data Connections
  - Composite.localhost.MyTDVDataSource
    - Tables
      - cache_status
      - cache_tracking
      - customers
      - employees
      - orderdetails
      - orders
      - orders_cache
      - pubtable_#copy#
      - shippingmethods
    - Stored Procedures
- Servers
- SharePoint Connections

Modifying or Deleting a Connection

A connection should be modified or deleted only if no active editor for its objects is opened. Otherwise your data could be lost.

To modify and delete a connection

1. Use the Visual Studio Server Explorer context menu for the corresponding node.
2. You can modify any of the settings by overwriting the existing values with new ones.

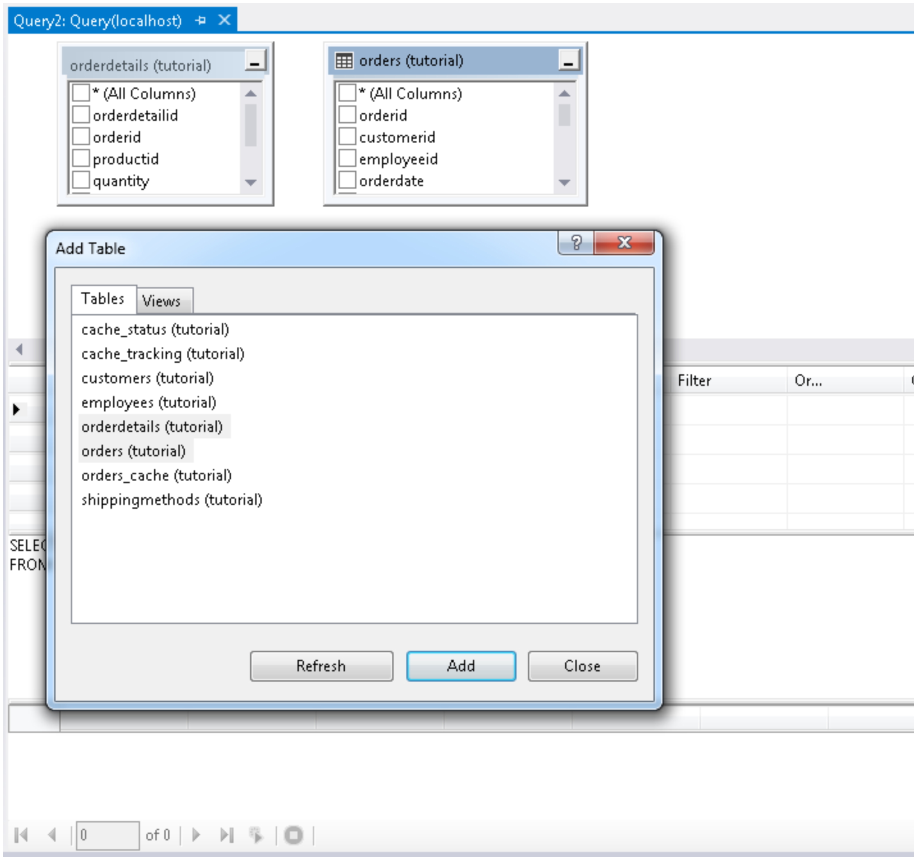
Working with the Server Explorer

Because you can customize the working area of Microsoft Visual Studio, the views presented in the screen shots below might differ from what you see. From the Server Explorer panel a table editor can be launched to create new queries from published TDV objects (tables, views and procedures).

To work with published TDV objects in Visual Studio

1. From the Visual Studio Server Explorer pane select and expand a TDV data source node.
2. Right-click on the Tables node and select New Query.
3. In the Add Table dialog, select the tables you want in the query and click Add, then Close the dialog.

4. Use the Query Designer to query the TDV data sources in Visual Studio.



Working with the Visual ToolBox Items

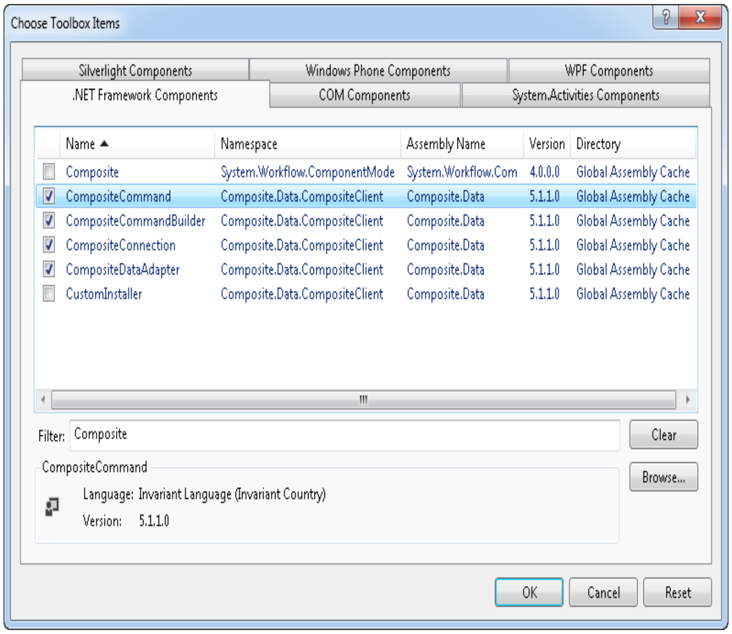
TDV ADO.NET components can be added to the Visual Studio .NET Toolbox. After the objects are in the toolbox, Visual Studio Designer lets you add objects to the Windows Forms so that then you can configure them using either the Properties windows or a wizard. The following components are available:

- CompositeConnection
- CompositeCommand
- CompositeCommandBuilder
- CompositeDataAdapter

This is only available for Windows Forms applications (not ASP.NET).

To use the ToolBox items

1. From the Visual Studio Tools menu, select Choose Toolbox Items.
2. Enter “Composite” in the Filter field. The following items appear.



	Name	Namespace	Assembly Name	Version	Directory
☐	Composite	System.Workflow.ComponentMode	System.Workflow.Com	4.0.0.0	Global Assembly Cache
☑	CompositeCommand	Composite.Data.CompositeClient	Composite.Data	5.1.1.0	Global Assembly Cache
☑	CompositeCommandBuilder	Composite.Data.CompositeClient	Composite.Data	5.1.1.0	Global Assembly Cache
☑	CompositeConnection	Composite.Data.CompositeClient	Composite.Data	5.1.1.0	Global Assembly Cache
☑	CompositeDataAdapter	Composite.Data.CompositeClient	Composite.Data	5.1.1.0	Global Assembly Cache
☐	CustomInstaller	Composite.Data.CompositeClient	Composite.Data	5.1.1.0	Global Assembly Cache

Filter: Composite

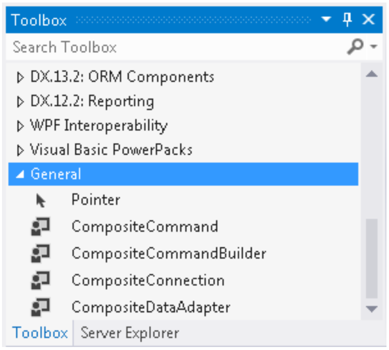
CompositeCommand

Language: Invariant Language (Invariant Country)

Version: 5.1.1.0

3. Check the check boxes next to the items you want to add as shown above. Visual Studio displays the toolbox items added.
4. Click OK.
5. Create or open an existing Windows Forms project.
6. Display the Toolbox Components.

7. Notice that the Composite components appear in the Toolbox panel rolled under the General tab.



8. Drag-and-drop the TDV items to your project.

9. Use the Properties panel to configure the TDV components.

Defining an ADO.NET Client using a Connection URL

It is possible to create client program and establish a connection to your data through TDV without having to define a DSN.

Note: The following instruction are guidelines only.

This topic also includes the following:

- ADO.NET Driver Connection URL Properties, page 160

To create a client program without defining a DSN connection

1. Create and declare your connection URL, using the following syntax:

```
{Driver=<driver name>;Server=<fully qualified
hostname>;Port=<port>;User=<username>;Password=<password>;domain=<
domain name>;dataSource=<datasource name>
```

The following examples show how the syntax might be implemented in a C++ program.

Platform	Example
Windows	`SQLCHAR dsn[] = "Driver={TDV8.0};Server=localhost;Port=9401;User=admin;Password=admin;Domain=composite;dataSource=redwood;user=admin;password=admin;validateRemoteHostname=false;connectTimeout=3000;enableFastExec=false";`

For other URL properties, see ADO.NET Driver Connection URL Properties, page 160.

2. Declare the user name and password variables for the connection statement.
3. (Optional) Determine the ADO.NET driver name using one of the following methods.

Platform	Location of Name
Windows	The driver name can be found in the Add/Remove Programs of the Windows control panel.

4. (Optional) Write a small sample program to test the connection URL.
5. Create or modify your client program so that it includes the connection syntax. For example, you must include a statement similar to the following to establish the connection:

```
conn = DriverManager.getConnection(url, userName, password);
```

ADO.NET Driver Connection URL Properties

This table lists the names of properties that you can specify in the ADO.NET connection URL.

ADO.NET Property	Description
alternatesecuritycredentials	Specifies an alternate security property value to the identity within the current session. This is used to allow the user passing security property to the data source. **Note**: You may get unexpected results when multiple requests are made on the same session or when multiple identities access the same session.

ADO.NET Property	Description
caseSensitive	Specifies case sensitivity in the request values. By default (`false`), requests are not case-sensitive.
catalog	Specifies the catalog that is used for the connection.
commitFailure	Specifies the behavior if commit failed. Possible values are: rollback or bestEffort.
commitInterrupt	Specifies behavior if commit is interrupted.Possible values are: ignore, log, fail.
compensate	Specifies correcting behavior. If enabled, compensation blocks will be run if the transaction rolls back. Possible values: disabled or enabled. Default value is disabled.
connectTimeout	Time-out for initial connection, in seconds. Use 0 (zero) for infinite time-out.
dataSource	Specifies the data source that is used for all connections.
domain	Specifies an identification string that defines a realm of administrative autonomy, authority, or control.
enableFastExec	Valid values are true and false. The default value is false. Results are processed and returned immediately (instead of a round trip) when a query is submitted, potentially improving performance of low latency queries.
enableFlood	Values are true or false. Default value is true. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.
enableReconnectOnError	Specifies cluster reconnection behavior.
encrypt	True or False. Used for SSL security. Set to True to enable SSL security. The default is False.
errorloggingenabled	Enable ADO.NET driver log.

ADO.NET Property	Description
fetchBytes	Maximum number of rows to fetch for a batch based on batch size, in bytes. Setting fetchBytes to a very large number can cause an Out Of Memory error in the server. The value set for fetchBytes affects the memory used on the client and the TDV server, so the value should be set based on the heap size configured. The default value is used if this property is set to zero.
fetchRows	Maximum number of rows to fetch for a batch. The default value is used if this property is set to zero.
host/server	TDV Server host name.
ignoreTrailingSpace	Ignore trailing spaces at the end of values. Default: **`false`**.
integrated_authentication	Specified the integrate authentication method with three values: '(Disabled)', 'Kerberos', 'NTLM'
kerberos spn	Valid on Windows platform only, not useful on UNIX platforms. Kerberos SPN value, only useful if the SSO value equals Kerberos.
keyStoreFile	Used for SSL security. Specifies the keystore file to use for verification. The file is in the PKCS#12 format. The default keystore could be found at apps/ADO.NET/ Security/cis_ado_keystore.pfx, which includes the client certificate and private key.
keyStorePass	Used for SSL security. Specifies the password for the KeyStoreFile. The default value is 'changeit'.
locale	Value that defines the user's language and country.
maxpoolsize	Sets the maximum number of connections that are opened in the same pool at the same time. If the maximum is reached and no usable connection is available, subsequent requests are queued until a connection is available.
minpoolsize	Sets the minimum number of connections that is maintained even if inactive to avoid the time cost of recreating new connections for a new request.
nometadata	Blocks return of result-set metadata during query execution.

ADO.NET Property	Description
paramMode	Controls the behavior of OUT parameters for stored procedures: • normal—Report OUT parameters in procedure metadata as OUT parameters. • return—Report OUT parameters as return values. • omit—Omit OUT parameters from metadata. • omitCursors—Omit output cursors from metadata.
password/pwd	Specifies the password for the user name that you specify in the Username property. These values are used for your data source connection.
pingInterval	Maximum time to wait before sending a ping request while waiting for a result from TDV, in seconds.
pingTimeoutWindow	The length of time the JDBC or ODBC client waits before closing a connection to the TDV server, after a ping to the TDV server has failed. The value of this parameter should be greater than or equal to the "PingInterval" parameter. If a ping sent to the TDV server fails, the ODBC or JDBC client continues to send pings to TDV to check status. If these client pings continue to fail after the TimeoutWindow has expired, the ODBC or JDBC client closes the connection to the TDV server and sends a message. While the TimeoutWindow has not expired, the ODBC or JDBC client connection stays open and continues to send pings to the TDV server waiting for a response. The default for this property is 0, which means the setting is not being used.
pooling	When true, inactive connections are saved and reused as necessary.
port	TDV Server listening port.
registerOutputCursors	• true—Bind or register output cursors as output parameters. • false—Do not bind or register output cursors as output parameters; instead, use SQLMoreResults to access the cursors.
requestTimeout	Time-out for query commands and other requests
sessionTimeout	Session inactivity timeout, in seconds. Set to zero for infinite timeout.

ADO.NET Property	Description
sessionToken	Uses the URL to set a session token value for client authorization when using TDV with a client restricted license. Example: &sessionToken=<VALUE>
spn	Valid on Windows platform only, not useful on UNIX platforms. Kerberos SPN value, only useful if the SSO value equals Kerberos.
sso	Valid on Windows platform only, not useful on UNIX platforms. Single-sign-on type: ""/(Disabled), Kerberos or NTLM. The default value is "", which forces the client application to provide user and password information to connect.
statusinterval	Log thread and connection status with a specified time interval in seconds.
stripDuplicates	Values are true or false. Default value is false. If true, the server will detect for duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire. This would potentially lead to data savings across the wire.
stripTrailingZeros	Determines whether decimal result values are to be returned with trailing zeros removed.
traceFolder	Absolute directory to save trace file, the trace file name is "CsOdbcDebug_"+<DSN Name>+".log". the default folder is C:\ or $COMPOSITE_HOME
traceLevel	Valid values are off, fatal, error (this is the default), debug, warn, info, debug, and all. The valid values for client-side log settings are off, fatal, error (default), warn, info, debug, all, stdout. On UNIX-based platforms, the log file CsOdbcDebug.log is created in the directory specified by the environment variable COMPOSITE_HOME.
user_tokens	Authentication values that can be packaged for delivery.
user/uid	Specifies the user name for connections to the data source.

ADO.NET Property	Description
validateRemoteCert	Useful on Windows platform only, it is ignored on UNIX platforms. When true, the TDV client initiates handshake validation, validating the TDV certificate and using it for password encryption. If validation fails, no connection is established. When false (default), no certificate validation is performed prior to the establishment of a connection. The TDV Server certificate is loaded from the Truststore File Location set in the Studio Configuration panel. The Keystore Key Alias is used when it is configured for use. For more information, refer to “TDV Configuration Parameters” in the *TDV Administration Guide*. The TDV ADO.NET driver uses the system certification store to validate the certificate. The TDV Server certificate must be added to this client trust store or validation fails.
validateRemoteHostname	Useful on Windows platform only, it is ignored on UNIX platforms. When true, the ADO.NET driver compares the value of host in the URL with the subject CN (common name) value in the certificate received from the targeted TDV Server. If the host name validation fails, the connection is not established. When false (default), the host name validation is not performed.
AccessToken	The authorization tokens used for OAuth2 authentication. The tokens are represented in a specific format - <header>.<payload>.<signature>. Each of the parts of the token is in a JSON format. The access token is used in place of id/password credentials, with a limited lifetime & privileges.
AccessTokenType	The type of the AccessToken. JWT (JSON Web Token) is the default supported format. JWT token is a self-contained JSON form and ideal for federation.

Sample Code for Testing of an ADO.NET Driver

The examples in this section describe various way to write code to consume TDV resources through an ADO.NET connection interface. This sample code was developed and tested on a Microsoft Windows XP Professional platform, using Microsoft Visual Studio and TDV.

- Create a CompositeConnection Object, page 166
- Create a CompositeCommand Object, page 168
- Select Data from a TDV Published Resource, page 169
- Getting the Column Type, page 171
- Getting Column Metadata, page 172
- Using an Update Operation in the Sample Code, page 173
- About Using Parameters, page 174
- Invoking a Stored Procedure Example, page 177
- Example with Special Data Types, page 182
- Retrieving Metadata, page 184
- Retrieving Tables with a Named Schema, page 185

Create a CompositeConnection Object

This creates a CompositeConnection object with a parameter object called CompositeConnectionStringBuilder. When it has been created, you must call the open method to connect to the server. If there is an exception when connecting to the server, the sample code catch returns to try the connection again.

Always use the Close method to close the conn object when finished with it. Use the conn object to access the TDV Server.

To create a CompositeConnection object using compositeConnectionStringBuilder

1. Create a class called BaseTest. For example:

```
public class BaseTest
{

protected CompositeConnection conn;
   protected CompositeConnectionStringBuilder builder;
public BaseTest()
{
}
}
```

This code defines a constructor method.

2. Add code to create more examples.

   For example, build a CompositeConnectionStringBuilder object that requires a BuildConnectionString method to feed the object:

```
public class BaseTest
{
public BaseTest()
{
// Build the CompositeConnectionStringBuilder object when calling
the construction method.
BuildConnectionString();
}
// Construct the CompositeConnectionStringBuilder object.
private void BuildConnectionString()
   {
String connstring =
"host=localhost;port=9401;user=admin;password=admin;domain=composi
te;datasource=examples";
builder = new CompositeConnectionStringBuilder(connstring);
}
}
```

3. In the BaseTest class, create a CompositeConnection object, and create Open and Close methods and a CompositeConnection object with the following code:

```
// Create CompositeConnect
protected void Open()
{
try
   {
conn = new CompositeConnection(builder);
conn.Open();
}
catch (Exception ex)
{
throw new Exception(ex.Message);
}
}

protected void Close()
{
try
{
```

```
if (conn.State == ConnectionState.Closed)
return;
conn.Close();
}
catch (Exception ex)
{
Assert.Fail(ex.Message);
}
}

protected CompositeConnection GetConnection()
{
if (conn == null || conn.State == ConnectionState.Closed)
Open();
return conn;
}
```

Create a CompositeCommand Object

The following sections provide code samples for how to select or update data from a published TDV resource. You can use multiple programmatic styles to access, use, and change the data.

- Using a SQL Statement to Create the CompositeCommand Object, page 168
- Using the conn.CreateCommand Method to Create the CompositeCommand Object, page 169
- Using CompositeCommand to Create the CompositeCommand Object, page 169

Using a SQL Statement to Create the CompositeCommand Object

The following builds a CompositeCommand object with a SQL statement and the CompositeConnection object.

```
try
{
String sql = "delete from products where ProductID=1111";
   CompositeCommand cmd = new CompositeCommand(sql, conn);
}
catch (Exception ex)
{
Throw ex;
}
```

Using the conn.CreateCommand Method to Create the CompositeCommand Object

The following retrieves a CompositeCommand object by calling the conn.CreateCommand method. Set the SQL statement to cmd before using it to access TDV.

```
try
{
CompositeCommand cmd = conn.CreateCommand();
cmd.CommandText = "delete from products where ProductID=1111";
}
catch (Exception ex)
{
Throw ex;
}
```

Using CompositeCommand to Create the CompositeCommand Object

Use the following to create a new object. Call the CompositeCommand default constructor and set the CommandText and Connection objects before using it.

```
try
{
CompositeCommand cmd = new CompositeCommand();
cmd.CommandText = "delete from products where ProductID=1111";
cmd.Connection=conn;
}
catch (Exception ex)
{
Throw ex;
}
```

Select Data from a TDV Published Resource

You can select the data from the TDV Server with a SQL statement such as:

```
select ProductName from products where ProductID=1111
```

To select data from a TDV published resource

1. Edit the following code to send the SQL:

```
public void TestExecuteScalar()
{
CompositeConnection conn = GetConnection();
try
{
CompositeCommand cmd = new CompositeCommand(); cmd.CommandText =
"select ProductName from products where ProductID=1111";
cmd.Connection = conn;
String name = (String)cmd.ExecuteScalar();
Console.WriteLine("(ProductName:" + name + ",TestExecuteScalar)");
}
catch (Exception ex)
{
throw ex;
}
}
```

2. Call the cmd.ExecuteScalar method to get the object that locates the first row and first column in the result set of the ExecuteScalar.
3. Conversion to a recognized data type is necessary, because it is an Object type value.

Select Data from a TDV Published Resource on the Server

Here is another example illustrating a data selection from the server.

```
public void TestExecuteReader()
{
try
{
CompositeCommand cmd = new CompositeCommand("select
ProductID,ProductName from products where
ProductID=1111",GetConnection());
CompositeDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
int ProductID = reader.GetInt32(0);
String ProductName = reader.GetString("ProductName");
Console.WriteLine("(ProductID:" + ProductID + ",ProductName:" +
ProductName + "TestExecuteReader)");
}
reader.Close();
}
catch (Exception ex)
{
throw ex;
}
}
```

To select data from a published resource on the server

1. Call cmd.ExecuteReader to get a CompositeDataReader object.
2. Access result set data using the CompositeDataReader object. The Read method tells you whether there is row data. If the Read method result value is

true, columns are accessible by their respective ordinal integer or by column name. The ordinal integer numbering begins with 0, not with 1.

3. To get the first column value of the current row, use the statement:

```
int ProductID = reader.GetInt32(0);
```

4. Because the object in that column is an integer, the GetInt32 method retrieves the value. If the column type were a string, the GetString (column name | column ordinal) method would retrieve the value.
5. Close the reader using reader.Close().

   If the reader is not closed, errors occur.

Getting the Column Type

This topic provides sample code showing how to get the column type.

To get the column type

1. Use the following sample code to get the column type:

```
public void TestColumnType()
{
CompositeConnection conn = GetConnection();
try
{
CompositeCommand cmd = conn.CreateCommand();
cmd.CommandText = "select * from products where ProductID=1111";
CompositeDataReader reader = cmd.ExecuteReader();
int columns = reader.FieldCount;
//reader.Read();
for (int i = 0; i < columns; i++)
{
Console.WriteLine("field name:" + reader.GetName(i) + ",field
type:" + reader.GetFieldType(i));
}
reader.Close();
}
catch (Exception ex)
{
throw ex;
}
}
```

   The number of columns you can access is given by reader.FieldCount. If the value is 3, three columns are in the select result, so you can access columns 0, 1, and 2.

2. Call the GetName method to get the column name.
3. Call GetFieldType to get the column type.

Getting Column Metadata

The following code shows a way to access column metadata using CompositeDataReader.

You can call GetSchemaTable to get the DataTable object. It contains some rows, and returns every row present with column metadata. The metadata contains: `ColumnName, ColumnOrdinal, ColumnSize, NumericPrecision, NumericScale, DataType, ProviderType, IsLong, AllowDBNull, IsReadOnly, IsRowVersion, IsUnique, IsKey, IsAutoIncrement, BaseSchemaName, BaseCatalogName, BaseTableName, and BaseColumnName.`

To get the column metadata

1. Use the following code to retrieve the rows and the metadata by name. The BaseSchemaName, BaseCatalogName, and BaseTableName are always present.

```
public void TestReaderMetadata()
{
CompositeConnection conn = GetConnection();
Try
{
CompositeCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM PRODUCTS WHERE ProductID=1111";
CompositeDataReader reader = cmd.ExecuteReader();
DataTable dt = reader.GetSchemaTable();
INT ROWS = dt.Rows.Count;
IF (rows > 0)
{
foreach (DataRow row in dt.Rows)
{
String ColumnName = (String)row["ColumnName"];
int ColumnOrdinal = (int)row["ColumnOrdinal"];
int ColumnSize = (int)row["ColumnSize"];
int NumericPrecision = (int)row["NumericPrecision"];
int NumericScale = (int)row["NumericScale"];
Type DataType = (Type)row["DataType"];
int ProviderType = (int)row["ProviderType"];
bool IsLong = (bool)row["IsLong"];
bool AllowDBNull = (bool)row["AllowDBNull"];
bool IsReadOnly = (bool)row["IsReadOnly"];
bool IsRowVersion = (bool)row["IsRowVersion"];
bool IsUnique = (bool)row["IsUnique"];
bool IsKey = (bool)row["IsKey"];
bool IsAutoIncrement = (bool)row["IsAutoIncrement"];
String BaseSchemaName = (String)row["BaseSchemaName"];
String BaseCatalogName = (String)row["BaseCatalogName"];
String BaseTableName = (String)row["BaseTableName"];
String BaseColumnName = (String)row["BaseColumnName"];
Console.WriteLine("Column properties:");
Console.WriteLine("ColumnName:" + ColumnName+
",ColumnOrdinal:" + ColumnOrdinal+
",ColumnSize:" + ColumnSize+
",NumericPrecision:" + NumericPrecision+
",NumericScale:" + NumericScale+
",DataType:" + DataType+
",ProviderType:" + ProviderType+
",IsLong:" + IsLong+
",AllowDBNull:" + AllowDBNull+
",IsReadOnly:" + IsReadOnly+
",IsRowVersion:" + IsRowVersion+
",IsUnique:" + IsUnique+
",IsKey:" + IsKey+
",IsAutoIncrement:" + IsAutoIncrement+
",BaseSchemaName:" + BaseSchemaName+
",BaseCatalogName:" + BaseCatalogName+
",BaseTableName:" + BaseTableName+
",BaseColumnName:" + BaseColumnName+"."
);
}
}
reader.Close();
}
catch (Exception ex)
{
Assert.Fail(ex.Message);
}
}
```

Using an Update Operation in the Sample Code

Update operations can perform insertions, updates, and deletions of data and rows. This example shows how to run update SQL. The most important method is ExecuteNonQuery. Usually it is used to execute update SQL. The return value is the number of rows that are affected by the update.

```
public void TestUpdate()
{
CompositeConnection conn = GetConnection();
try
{
CompositeCommand cmd = new CompositeCommand("delete from products
where ProductID=1111", conn);
int cnt = cmd.ExecuteNonQuery();
cmd.Parameters.Clear();
cmd.CommandText = "insert into
products(ProductID,ProductName,ProductDescription)
values(1111,'Composite DataBase','big base.')";
cnt = cmd.ExecuteNonQuery();
cmd.Parameters.Clear();
cmd.CommandText = "update products set ProductName='TDV' where
ProductID=1111";
cnt=cmd.ExecuteNonQuery();
cmd.CommandText = "select ProductName from products where
ProductID=1111";
string name = (string)cmd.ExecuteScalar();
if (!name.Equals("TDV"))
Console.WriteLine("error occurred.");
}
catch (Exception ex)
{
throw ex;
}
}
```

About Using Parameters

The construct method in the following sample code is for CompositeParameter. You can create a new parameter with the methods:

```
public CompositeParameter(string parameterName, object value)
public CompositeParameter(string parameterName, object value,
CompositeDbType dbType)
public CompositeParameter(string parameterName, CompositeDbType
dbType)
```

- The current argument name is parameterName. This must not be empty.
- The value of the current argumentvalue.
- The type to assign to the current parameter is dbType. The value of dbType must be one of the CompositeDbType objects.

Methods for Adding Parameters

There are several ways to add parameters to the CompositeParameterCollection object. In the following sample, Method A uses a prepared SQL statement as the cmd.CommandText. Method B illustrates the binding of a placeholder with a parameter. This method can be used except when working with the following data types: Clob, Date, Time, and Timestamp.

```
((CompositeParameterCollection)cmd.Parameters).Add("@ProductID",
1111);
```

Convert the cmd.Parameters object to the CompositeParameterCollection type. After converting the Add method to declare a new parameter name:
`'public CompositeParameter Add(string parameterName, object value)'`

Method C uses the following JDBC style SQL statement:
`update products set ProductName=? where ProductID=?`

The question mark (?) is a placeholder without an appended name string. The placeholder is bound with the corresponding parameter as follows:

```
cmd.Parameters.Add(new CompositeParameter("?ProductName", "TDV"));
cmd.Parameters.Add(new CompositeParameter("?ProductID", 1111));
```

The public CompositeParameter Add(CompositeParameter value) method is used to add a parameter. The first parameter is bound to the first placeholder, and the last parameter is bound to the last placeholder.

The following sample code contains four possible methods (A through D) for implementing parameters. The sample uses the ADO.NET placeholder characters of ? and @.

```
public void TestParameter()
{
CompositeConnection conn = GetConnection();
try
{
//Method A
CompositeCommand cmd = conn.CreateCommand();
cmd.CommandText = "delete from products where
ProductID=@ProductID";
cmd.Parameters.Add("@ProductID", CompositeDbType.INTEGER);
cmd.Parameters[0].Value = 1111;
int cnt = cmd.ExecuteNonQuery();
//Method B
cmd.Parameters.Clear();
cmd = new CompositeCommand("insert into
products(ProductID,ProductName) values(@ProductID,@ProductName)",
conn);
((CompositeParameterCollection)cmd.Parameters).Add("@ProductID",
1111);
((CompositeParameterCollection)cmd.Parameters).Add("@ProductName",
"Discovery");
cnt = cmd.ExecuteNonQuery();
//Method C
cmd.Parameters.Clear();
cmd.CommandText = "update products set ProductName=? where
ProductID=?";
cmd.Parameters.Add(new CompositeParameter("?ProductName", "TDV"));
cmd.Parameters.Add(new CompositeParameter("?ProductID", 1111));
cnt = cmd.ExecuteNonQuery();
//Method D
cmd.Parameters.Clear();
cmd.CommandText = "select ProductName from products where
ProductID=?ProductID";
cmd.Parameters.Add(new CompositeParameter("?ProductID", 1111));
String ProductName = (String)cmd.ExecuteScalar();
if (!ProductName.Equals("TDV"))
Console.WriteLine("error happen.");
}
catch (Exception ex)
{
throw ex;
}
}
```

About ADO.NET Placeholders

The following is a sample SQL statement with a parameter that uses a placeholder:

```
delete from products where ProductID=@ProductID
```

In the example, **`@ProductID`** is the placeholder argument name, and the @ and ? characters are placeholders. Using ? is recommended.) The argument name must not be empty.

If the SQL statement is a prepared statement, you must bind @ProductID with a parameter object. This object contains the argument name, value and type. ADO.NET can send those values to the server to gain access to the result.

From Method A in About Using Parameters, page 174, a prepared SQL statement was used as the cmd.CommandText:

```
cmd.CommandText = "delete from products where
ProductID=@ProductID";
```

You can bind the placeholder in the following manner:

```
cmd.Parameters.Add("@ProductID", CompositeDbType.INTEGER);
```

The cmd.Parameters is an object of the CompositeParameterCollection and every CompositeCommand object has a cmd.Parameters object that contains all parameters bound to the all placeholder.

You could instead use the cmd.Parameters.Add method to bind a placeholder and parameter object. Adding a parameter object to the @ProductID placeholder and defining its type as a CompositeDbType.INTEGER requires a value of the parameter like the following:

```
cmd.Parameters[0].Value = 1111;
```

The cmd.Parameters[0] refers to the first parameter object, with a value of 1111.

Invoking a Stored Procedure Example

After all definitions are in place, the call to ExecuteNonQuery invokes the stored procedure. The LookupProduct procedure (in the TDV directory path ~/shared/examples/LookupProduct) must be published before this sample code is executed; otherwise, the return value of ExecuteNonQuery is not valid.

The procedure SQL statement is the same for JDBC and ODBC.

The sample gets a CompositeCommand object, and then defines the CommandType as a StoredProcedure using the following line::

```
cmd.CommandType = CommandType.StoredProcedure;
```

An exception can occur if the CompositeCommand object is called without this specification.

A new ID parameter is added with the CompositeParameter object type:

```
CompositeParameter id = new CompositeParameter("?ProductID", 12,
CompositeDbType.INTEGER);
```

The CompositeParameter ID parameter object gets a name, value, and type:

```
"?ProductID", 12, and CompositeDbType.INTEGER.
```

Procedure parameters must have a specified direction. There are four parameter directions in the ADO.NET standard:

- Input
- Output
- InputOutput
- ReturnValue

In this example, ProductID is an input parameter, so the example code defines:

```
id.Direction = ParameterDirection.Input;
```

The example also has an output parameter that is declared with the following line:

```
CompositeParameter cursor = new CompositeParameter("?cursor",
CompositeDbType.OTHER);
```

The name is set to a ?cursor placeholder of type CompositeDbType.OTHER. When the parameter is a cursor, you must specify CompositeDbType.OTHER as the data type. Setting the direction is required:

```
cursor.Direction = ParameterDirection.Output;
```

The sample code adds those parameter objects with the lines:

```
cmd.Parameters.Add(id);
cmd.Parameters.Add(cursor);
```

To read a valid output value the sample uses:

```
CompositeDataReader reader = (CompositeDataReader)cursor.Value;
```

Cursor output must be fetched by the CompositeDataReader. All cursor data is mapped to the CompositeDbType.OTHER data type, so returned values are CompositeDataReader objects. The CompositeDataReader object can be obtained from cursor.Value.

The following sample code illustrates one way to invoke a procedure:

```
public void TestSelect()
{
CompositeConnection conn = GetConnection();

String sql = "{call LookupProduct(?ProductID,?cursor)}";
CompositeCommand cmd = new CompositeCommand(sql, conn);
cmd.CommandType = CommandType.StoredProcedure;
CompositeParameter id = new CompositeParameter("?ProductID", 12,
CompositeDbType.INTEGER);
id.Direction = ParameterDirection.Input;
CompositeParameter cursor = new CompositeParameter("?cursor",
CompositeDbType.OTHER);
cursor.Direction = ParameterDirection.Output;
cmd.Parameters.Add(id);
cmd.Parameters.Add(cursor);
try
{
cmd.ExecuteNonQuery();
CompositeDataReader reader = (CompositeDataReader)cursor.Value;
if (reader != null)
{
String ProductName;
int ProductID;
String ProductDescription;
if (reader.Read())
{
//ProductName=reader.GetString("ProductName");
ProductName = (String)reader["ProductName"];
//ProductID=reader.GetInt32("ProductID");
ProductID = (int)reader["ProductID"];
//ProductDescription=reader.GetString("ProductDescription");
ProductDescription = (String)reader["ProductDescription"];
}
}
}
catch (Exception ex)
{
throw ex;
}
}
```

The following sample code defines a method with a SQL string, SQL, that can be used to call a stored procedure, p_integer, with a placeholder parameter, ?int, bound to intValue.

The definition of the CompositeParameter is the following:

```
CompositeParameter intValue = new CompositeParameter("?int",
CompositeDbType.INTEGER);
```

The ?int is an InputOutput parameter, with a type of CompositeDbType.INTEGER.

Set a procedure parameter value with a code line like:

```
intValue.Value = 12
```

Retrieve a procedure parameter value using a line like:

```
int value = (int)intValue.Value
```

The following code sample is another way to invoke a stored procedure:

```
public void TestInOut()
{
string sql = "{call p_integer(?int)}";
CompositeCommand cmd = new CompositeCommand(sql, conn);
cmd.CommandType = Command60
Type.StoredProcedure;
CompositeParameter intValue = new CompositeParameter("?int",
CompositeDbType.INTEGER);
cmd.Parameters.Add(intValue);
intValue.Direction = ParameterDirection.InputOutput;
intValue.Value = 12;
try
{
```

```
cmd.ExecuteNonQuery();
int value = (int)intValue.Value;
}
catch (Exception ex)
{
throw ex;
}
}
```

Using CompositeCommandBuilder

The CompositeCommandBuilder can be used to execute code to access or modify TDV resources. The following example code deletes the first row of the current SELECT. The builder.GetUpdateCommand is necessary when using a Microsoft implementation, but is optional in the following sample.

To use the CompositeCommandBuilder

1. Define a CompositeDataAdapter object with a SQL statement and connection object.
2. Create a new CompositeCommandBuilder object, with a CompositeDataAdapter object as an argument.
3. Create a new DataTable object and clear it. Use a call to the da.Fill(dt) method to populate it with data.
4. To delete the first row, make this call: dt.Rows[0].Delete(). The deletion is finalized after calling the da.Update(dt) statement.

```
public void TestDelete()
{
CompositeConnection conn = GetConnection();
CompositeDataAdapter da = new CompositeDataAdapter("select * from
products  where ProductID in (1111)", conn);
CompositeCommandBuilder cb = new CompositeCommandBuilder(da);

DataTable dt = new DataTable();
dt.Clear();
da.Fill(dt);

dt.Rows[0].Delete();
da.Update(dt);
}
```

5. Insert rows into a cursor with specified values. Section A marks where the insertion of a new row begins.

```
public void TestInsert()
{
try
{
CompositeConnection conn = GetConnection();
CompositeDataAdapter da = new CompositeDataAdapter("select * from
products", conn);
CompositeCommandBuilder cb = new CompositeCommandBuilder(da);
DataTable dt = new DataTable();
da.Fill(dt);

//Section A
//Create the new row
DataRow row = dt.NewRow();
//The values in the new row are set with:
row["ProductID"] = 1111;
row["ProductName"] = "TDV";
row["ProductDescription"] = DBNull.Value;
dt.Rows.Add(row);
//Insertion of the new rows into the database
da.Update(dt);
}
catch (Exception ex)
{
throw ex;
}
}
```

6. (Optionally) When using a Microsoft implementation, you must use the builder.GetUpdateCommand method. Here is some sample code from a Microsoft-based implementation:

```
public DataSet TestUpdate()
{
CompositeConnection conn = GetConnection();
String queryString = "select * from products where ProductID in
(1111)";
String tableName = "products";
CompositeDataAdapter adapter = new CompositeDataAdapter();
adapter.SelectCommand = new CompositeCommand(queryString, conn);
CompositeCommandBuilder builder = new
CompositeCommandBuilder(adapter);

DataSet dataSet = new DataSet();
adapter.Fill(dataSet, tableName);
dataSet.Tables[0].Rows[0]["ProductName"] = "Discovery";

builder.GetUpdateCommand();
adapter.Update(dataSet, tableName);
return dataSet;
}
```

Example with Special Data Types

This section contains several code samples that show different ways to define and use special data types, including CLOB, date, time, and datetime.

The following is procedure invoking code that uses the time data types.

```
public void TestTime()
{
string sql = "{call p_time(?time1,?date,?timestamp)}";
CompositeCommand cmd = new CompositeCommand(sql, conn);
cmd.CommandType = CommandType.StoredProcedure;
CompositeParameter timestamp = new CompositeParameter("?timestamp",
new CompositeTimeStamp(DateTime.Parse("2003-12-24
03:12:52.112")));
CompositeParameter time = new CompositeParameter("?time1", new
CompositeTime(DateTime.Parse("03:16:54.111")));
CompositeParameter date = new CompositeParameter("?date",
DateTime.Parse("1999-09-11"), CompositeDbType.DATE);
timestamp.Direction = ParameterDirection.InputOutput;
time.Direction = ParameterDirection.InputOutput;
date.Direction = ParameterDirection.InputOutput;
cmd.Parameters.Add(time);
cmd.Parameters.Add(date);
cmd.Parameters.Add(timestamp);
try
{
cmd.ExecuteNonQuery();
string timeStr = ((DateTime)time.Value).ToString("HH:mm:ss.fff");
string dateStr = ((DateTime)date.Value).ToString("yyyy-MM-dd");
string timestampStr =
((DateTime)timestamp.Value).ToString("yyyy-MM-dd HH:mm:ss.fff");
}
catch (Exception ex)
{
throw ex;
}
}
```

If you want to use a date, time or timestamp data type, you must specify that data type or use a CompositeDate, CompositeTime, or CompositeTimestamp object.

```
CompositeParameter date = new CompositeParameter("?date",
DateTime.Parse("1999-09-11"), CompositeDbType.DATE)
```

You can create a new CompositeParameter object and set the type to one of the following:

```
CompositeDbType.DATE
CompositeDbType.TIME
CompositeDbType.TIMESTAMP
```

Set the **`value`** to CompositeTimeStamp, CompositeTime, or CompositeDate object.

If the DATE type parameter is not specified, the ADO.NET driver sets the type to TimeStamp by default.

Values retrieved as Date(time, timestamp) value are DateTime (.NET object) type. The GetDateTime method can be used to retrieve values that can be converted to DateTime objects.

```
CompositeParameter timestamp = new CompositeParameter("?timestamp",
new CompositeTimeStamp(DateTime.Parse("2003-12-24
03:12:52.112")));
```

Defining a CLOB type is shown in the following code; or you can explore using the CompositeClob type:

```
public void TestClob()
{
try
{
CompositeConnection conn = GetConnection();
string sql = "insert into ALL_TYPE(ID,COLUMN_10) values(?id,?clob)
";
CompositeCommand cmd = new CompositeCommand(sql, conn);
CompositeParameter id = new CompositeParameter("?id", 111);
CompositeParameter clob = new CompositeParameter("?clob",
"www.compositesw.com", CompositeDbType.CLOB);
cmd.Parameters.Add(clob);
cmd.Parameters.Add(id);
int cnt = cmd.ExecuteNonQuery();
}
catch (Exception e)
{
throw ex;
}
}
```

Retrieving Metadata

The metadata capabilities in TDV ADO.NET Driver are exposed through a generic API using the CompositeConnection object. The CompositeConnection object's GetSchema method has overloads that allow you to pass the name of the schema information, called the metadata collection, that you are interested in. You can also pass filter information. The following table shows the GetSchema overloads.

Overload	Description
GetSchema()	Gets a DataTable with a row for each metadata collection that is available with this provider. This option is the same as calling GetSchema("MetaDataCollections").
GetSchema(string)	Passes a metadata collection name and gets a DataTable containing a row for each item found in that metadata collection in the database.
GetSchema(string, string array)	Passes a metadata collection name and an array of string values that specify how to filter the results, and gets a DataTable containing a row for each filtered item found in the metadata collection in the database.

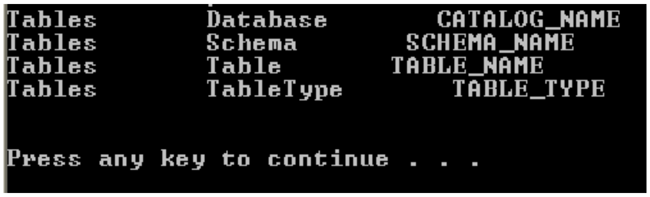
You need an open connection to execute the GetSchema method. You can start by calling the GetSchema method with no parameters. It returns a list of the available metadata collections.

The following is a sample that retrieves the restrictions information for TABLES.

```
public void TestRestrictionInfo()
{
string space = "       ";
CompositeConnection conn = GetConnection();
try
{
DataTable dt = conn.GetSchema("Restrictions");
foreach (DataRow myRow in dt.Rows)
{
if(myRow[0].ToString().ToUpper()  == "TABLES")
{
Console.WriteLine(myRow[0] + space + myRow[1] + space + myRow[2]);

}
}
}
catch (Exception ex)
{
throw ex;
}
}
```

After you run the program, the following results appear. It shows that the metadata TABLES have four restrictions: catalog_name, schema_name, table_name, and table_type.



```
Tables         Database       CATALOG_NAME
Tables         Schema       SCHEMA_NAME
Tables         Table       TABLE_NAME
Tables         TableType        TABLE_TYPE

Press any key to continue . . .
```

Retrieving Tables with a Named Schema

The following is an example of how to retrieve the tables where the schema name is a fixed value like Mytest.

```
public void TestGetTablesInfo()
{
CompositeConnection conn = GetConnection();
try
{
string[] res = new string[4];
res[1] = "Mytest";
res[3] = "Table";
DataTable dt = conn.GetSchema("Tables",res);
foreach (DataRow myRow in dt.Rows)
{
Console.WriteLine(myRow["table_name"]);
}
}
catch (Exception ex)
{
throw ex;
}
}
```

TIBCO ADO.NET 2020 Data Provider for TDV

Overview

The ADO.NET Provider for TIBCO(R) Data Virtualization offers the most natural way to access TDV data from .NET applications. The provider wraps the complexity of accessing TDV data in an easy-to-integrate, fully managed ADO.NET Data Provider.

The provider hides the complexity of accessing data and provides additional powerful security features, smart caching, batching, socket management, and more.

Key Features

- DataBind to TDV using Visual Studio wizards.
- Real-time access to TDV.
- Comprehensive support for create, read, update, and delete (CRUD) operations.

Getting Started

See Getting Started for A-Z guides on authenticating and connecting to TDV data. See the TDV integration guides for information on connecting from other applications.

Using ADO.NET

The provider has the same ADO.NET architecture as the native .NET data providers for SQL Server and OLEDB. Code with familiar classes such as CompositeConnection, CompositeCommand, CompositeDataAdapter, CompositeDataReader, CompositeDataSource, CompositeParameter, and so on. See Using ADO.NET for guides relating to these and other ADO.NET features such as batch processing, connection pooling, and calling stored procedures.

Entity Framework

You can leverage Entity Framework to quickly and easily model database resources using .NET objects. In Using ADO.NET (Entity Framework), you can find instructions related to EF6 setup and installation, as well as the creation of both model-first and code-first data models.

Entity Framework Core

Entity Framework Core provides a streamlined, cross-platform solution for modelling database resources as .NET objects. Using ADO.NET (Entity Framework Core) covers how to surface TDV data using EF Core console and ASP.NET applications. Additionally, you can find instructions for automatically building data models from data source metadata using reverse engineering (scaffolding).

LINQ

LINQ queries allow you to query and modify data using simple, strongly typed expressions. For instructions regarding the construction and usage of LINQ queries, see Using ADO.NET (LINQ).

SSRS

You can use the provider to enable real-time connectivity to TDV within your SSRS reports. Using ADO.NET (SSRS) details provider deploy, the creation of shared and embedded data sources and datasets, and the publishing of SSRS reports.

DbProviderFactory

The provider supports the creation of strongly typed DbProviderFactory and DBConnection objects in order to facilitate connecting to TDV with generic code. Using ADO.NET (DbProviderFactory) describes how to get connected as well as create, configure, and execute DbCommands.

Schema Discovery

See Schema Discovery to use standard ADO.NET schema collections to discover schema information and other metadata.

Connection String Options

The Connection properties describe the various options that can be used to establish a connection.

Getting Started

Connecting to TDV

Establishing a Connection shows how to authenticate to TDV and configure any necessary connection properties. You can also configure provider capabilities through the available Connection properties, from data modeling to firewall traversal. The Advanced Settings section shows how to set up more advanced configurations and troubleshoot connection errors.

Connecting from Visual Studio

The ADO.NET Provider for TIBCO(R) Data Virtualization provides a seamless integration with Microsoft Visual Studio. The provider is registered as an ADO.NET provider with Visual Studio, allowing for integration with visual designer tools, Server Explorer, and ADO.NET data source configuration wizards.

TDV Version Support

The provider enables read/write SQL-92 access to TIBCO(R) Data Virtualization version 7.0.1 and above.

Establishing a Connection

Connecting from Visual Studio

You can use Server Explorer to test the connection and explore the data source. To create the connection, right-click the Data Connections node in Explorer and click Add Connection. Select TIBCO TDV Data Source.

Set the Host, Domain, User, Password, and DataSource connection properties to connect to the TDV Server.

Using ADO.NET

This section provides a walk-through of writing data access code to TDV in ADO.NET.

Connecting from Code

See Establishing a Connection for the prerequisite information you need to deploy the provider and configure the connection to TDV. Connecting from Code shows how to connect with the classes CompositeConnection, CompositeConnectionStringBuilder, and, in ASP.NET, CompositeDataSource.

Discovering Schemas

You can use the classes detailed in Schema Discovery to discover the table schemas at run time. You can also query the available System Tables to retrieve schema information, data source information, and other data provider metadata.

Executing SQL

You can use native ADO.NET interfaces to execute data manipulation SQL to TDV: Querying with the DataReader and Querying with the DataAdapter provide code examples and guides to using native ADO.NET interfaces to access TDV data. Results can be processed from the DataTable instance filled or from the DataReader returned.

Updating the Data shows how to use the provider to update changes to a data set.

Connection Pooling

Instantiate pooled connections by configuring the connection string. See Connection Pooling to create and configure a pool.

Installed Assemblies

The assemblies shipped by the provider contain standard ADO.NET components, including components for creating ASP.NET applications and SSRS reports, from code and from the designer.

Choosing the Assemblies for Your Version of the .NET Framework

The lib folder in the installation directory has .NET 3.5 assemblies. The 4.0 subfolder contains .NET 4.0 assemblies that have been compiled with the 4.0 compiler. These will not work with older versions of the .NET Framework. To use the Entity Framework, you must compile with .NET 4.0. Finally, the netstandard2.0 subfolder contains the System.Data.CompositeClient assembly compiled with .NET Standard 2.0.

Determining Project Dependencies

The following sections list the main assemblies and assemblies you need to integrate with Visual Studio designers and other tools.

Main ADO.NET Assemblies

The ADO.NET Provider for TIBCO(R) Data Virtualization ships the following ADO.NET assemblies:

- System.Data.CompositeClient.dll: This is the main ADO.NET provider assembly.
- System.Data.CompositeClient.Designer.dll: This assembly contains design-time resources that you can include for a better development experience. It does not need to be deployed.

Entity Framework Assemblies

The provider supports Entity Framework with the following assembly:

- System.Data.CompositeClient.Entities.EF6.dll: This assembly includes support for Entity Framework 6 (EF6).

See Using ADO.NET (Entity Framework) for information on creating Entity Framework data models with the provider.

Entity Framework Core Assemblies

The provider supports EF Core with the following assembly:

- TIBCO.EntityFrameworkCore.CompositeClient.dll: This assembly includes support for Entity Framework Core.

See Using ADO.NET (Entity Framework Core) for information on creating EF Core data models with the provider.

SSRS Assemblies

The provider supports SSRS 2005 and above; the assemblies for each SSRS version are located in the SSRS subfolder in the installation folder. For example:

- TIBCO.SSRS2017.Composite.dll: This assembly is deployed to the report server.
- TIBCO.SSRS2017.Composite.Design.dll: This assembly contains design-time resources that you can include for a better development experience. It does not need to be deployed.

See Using ADO.NET (SSRS) for a guide to deploying the provider to your SSRS server and creating a report.

Connecting from Code

The ADO.NET Provider for TIBCO(R) Data Virtualization implements a standard DbConnection object in CompositeConnection. You can also use the CompositeConnectionStringBuilder to programmatically build, parse, and rebuild connection strings.

Creating Connection Objects

See Establishing a Connection for guides to defining the connection string and authenticating. Below is a typical invocation to create CompositeConnection objects.

C#

```
using (CompositeConnection connection =
  new
CompositeConnection("Host=myHost;Domain=myDomain;DataSource=myData
Source;User=myUser;Password=myPassword"))
{
  connection.Open();
}
```

VB.NET

```
Using connection As New
CompositeConnection("Host=myHost;Domain=myDomain;DataSource=myData
Source;User=myUser;Password=myPassword")
  connection.Open
End Using
```

Using CompositeConnectionStringBuilder

The following code example shows how to use an ADO.NET connection string builder to parse a connection string.

C#

```

CompositeConnectionStringBuilder builder =
  new
CompositeConnectionStringBuilder("Host=myHost;Domain=myDomain;Data
Source=myDataSource;User=myUser;Password=myPassword");
  //Pass the connection string builder an existing connection
string, and you can get and set any of the elements as strongly
typed properties.
```

```
  builder.ConnectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";
  //Now that the connection string has been parsed,
  // you can work with individual items:
  builder.MyString = "new property";
  builder.MyBoolean = true;

  // You can refer to connection keys using strings,
  // as well.
  builder["Logfile"] = "test.log";
  builder["Verbosity"] = 5;
```

VB.NET

```
Dim builder As CompositeConnectionStringBuilder = New
CompositeConnectionStringBuilder("Host=myHost;Domain=myDomain;Data
Source=myDataSource;User=myUser;Password=myPassword")
'Pass the connection string builder an existing connection string,
and you can get and set any of the elements using strongly typed
properties.
builder.ConnectionString =
Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;Pa
ssword=myPassword"
'Now that the connection string has been parsed,
' you can work with individual items:
builder.MyString = "new property"
builder.MyBoolean = True
' You can refer to connection keys using strings,
' as well.
builder("Logfile") = "test.log"
builder("Verbosity") = 5
```

Querying with the DataReader

The ADO.NET Provider for TIBCO(R) Data Virtualization implements two ADO.NET interfaces you can use to retrieve data from TDV: CompositeDataAdapter and CompositeDataReader objects. Whereas CompositeDataAdapter objects retrieve a single result set of all the data that matches a query, CompositeDataReader objects fetch data in subset increments as needed.

Using the CompositeDataReader

The CompositeDataReader retrieves data faster than the CompositeDataAdapter because it can retrieve data in pages. As you read data from the CompositeDataReader, it periodically requests the next page of results from the data source, if required. This causes results to be returned at a faster rate. The following example selects all the columns from the Products table:

C#

```
string connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection connection = new
CompositeConnection(connectionString)) {
  CompositeCommand cmd = new CompositeCommand("SELECT * FROM
Products", connection);

  CompositeDataReader rdr = cmd.ExecuteReader();

  while (rdr.Read()) {
    Console.WriteLine(String.Format("\t{0} --> \t\t{1}", rdr["Id"],
rdr["ProductName"]));
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using connection As New CompositeConnection(connectionString)
  Dim cmd As New CompositeCommand("SELECT * FROM Products",
connection)

  Dim rdr As CompositeDataReader = cmd.ExecuteReader()

  While rdr.Read()
    Console.WriteLine([String].Format(vbTab & "{0} --> " & vbTab &
vbTab & "{1}", rdr("Id"), rdr("ProductName")))
  End While
End Using
```

Updating the Data

Use the adapter's Update method to update the data. This overloaded method can take a DataTable as a parameter and will commit all the changes made to the data source. The name of the data table can be passed as an argument and can also be used to update the entire dataset in a more traditional manner. When using a data table as an argument to the Update method, the adapter evaluates the changes that have been made to the data table and executes the appropriate command for each row (INSERT, UPDATE, or DELETE).

The example below updates the ProductName of one of the Products entries.

C#

```
using (CompositeConnection connection = new
CompositeConnection(connectionString)) {
  CompositeDataAdapter dataAdapter = new CompositeDataAdapter(
    "SELECT Id, ProductName FROM Products", connection);

  dataAdapter.UpdateCommand = new CompositeCommand(
    "UPDATE Products SET ProductName = @ProductName " +
    "WHERE Id = @Id", connection);

  dataAdapter.UpdateCommand.Parameters.Add(new
CompositeParameter("@ProductName", DbType.String, "ProductName"));
  dataAdapter.UpdateCommand.Parameters.Add(new
CompositeParameter("@Id", DbType.String, "Id"));
  dataAdapter.UpdateCommand.Parameters[1].SourceVersion =
DataRowVersion.Original;

  DataTable table = new DataTable();
  dataAdapter.Fill(table);

  DataRow firstrow = table.Rows[0];
  firstrow["ProductName"] = "Ikura";

  dataAdapter.Update(table);

  Console.WriteLine("Rows after update.");

  foreach (DataRow row in table.Rows) {
    Console.WriteLine("{0}: {1}", row["Id"], row["ProductName"]);
  }
}
```

VB.NET

```
Using connection As New CompositeConnection(connectionString)
  Dim dataAdapter As New CompositeDataAdapter(
    "SELECT Id, ProductName FROM Products", connection)

  dataAdapter.UpdateCommand = New CompositeCommand(
    "UPDATE Products SET ProductName = @ProductName " +
    "WHERE Id = @Id", connection)

  dataAdapter.UpdateCommand.Parameters.Add(new
CompositeParameter("@ProductName", DbType.String, "ProductName"))
  dataAdapter.UpdateCommand.Parameters.Add(new
CompositeParameter("@Id", DbType.String, "Id"))
  dataAdapter.UpdateCommand.Parameters(1).SourceVersion =
DataRowVersion.Original

  Dim table As New DataTable()
  dataAdapter.Fill(table)

  Dim firstrow As DataRow = table.Rows(0)
  firstrow("ProductName") = "Ikura"

  dataAdapter.Update(table)

  Console.WriteLine("Rows after update.")

  For Each row As DataRow In table.Rows
    Console.WriteLine("{0}: {1}", row("Id"), row("ProductName"))
  Next
End Using
```

Using the CompositeDataSource

The CompositeDataSource enables you to use a single Web control to connect to TDV and query data. By binding the control to other controls such as the GridView or ListBox, you can display, edit, and save TDV data in an ASP.NET page. You can use the Visual Studio Designer to initialize the control, visually build queries, and bind controls; you can also define SQL commands from page code. The examples below show both approaches and use the GridView control as an example.

To use the CompositeDataSource, add a reference to System.Data.CompositeClient.Web.dll in your ASP.NET project.

Bind Data Programmatically

The following example shows how to bind the results of a TDV query to an ASP.NET data grid. After registering the assembly for use with the CompositeDataSource, set the DataSourceID field of the GridView control to the Id of the CompositeDataSource control:

```
<%@ Register Assembly="System.Data.CompositeClient.Web"
Namespace="System.Data.CompositeClient" TagPrefix="cc1" %>
...
<cc1:CompositeDataSource ID="CompositeDataSource1" runat="server"
ConnectionString="Host=myHost;Domain=myDomain;DataSource=myDataSou
rce;User=myUser;Password=myPassword" SelectCommand="SELECT * FROM
Products WHERE ProductName = 'Konbu' LIMIT
10"></cc1:CompositeDataSource>
<asp:GridView DataSourceID="CompositeDataSource1"
runat="server"></asp:GridView>
```

Bind Data Using the Designer

Complete the following steps to use the Designer in Visual Studio to bind the CompositeDataSource to a GridView control:

- Drag a GridView from the Toolbox onto the page.
- Click the Smart Tag of the GridView.
- Choose the option to create a new data source, which launches a wizard to configure the control.
- On the first page, select a connection string from the menu or click New Connection to define a new connection or to save a connection entry in the Web.config file.
- On the next page, select the option to define a SQL statement or visually build the query.
- When you exit the wizard, the GridView displays the columns of the result set.

Connection Pooling

The provider implements a standard ADO.NET connection pool. Set UseConnectionPooling to enable the pool. The following sections show how to configure and use them.

Working with Pooled Connections

Just as you would interact with a non-pooled connection, you use standard ADO.NET objects to get and close connections. But, in this case, the CompositeConnection object retrieved is a handle for the physical connection owned by the connection pool. When the connection is closed, instead of the connection being destroyed, the handle is returned to the pool, where it is available for the next connection request.

You must explicitly close the connection for it to be returned to the pool.

Creating Pooled Connections

The connection pools are based on the unique connection strings except for the Pool properties and any persisted temporary values. The following example instantiates three connections using two distinct connection pools. Two of the connections share the same connection pool, as they share the same connection string.

C#

```
using (CompositeConnection connection = new
CompositeConnection("Host=myHost;Domain=myDomain;DataSource=myData
Source;User=myUser;Password=myPassword")) {connection.Open();}

using (CompositeConnection connection = new
CompositeConnection("PoolIdleTimeout=-1;Host=myHost;Domain=myDomai
n;DataSource=myDataSource;User=myUser;Password=myPassword"))
{connection.Open();}

using (CompositeConnection connection = new
CompositeConnection("Timeout=15;Host=myHost;Domain=myDomain;DataSo
urce=myDataSource;User=myUser;Password=myPassword"))
{connection.Open();} //spawns a new pool
```

VB.NET

```
Using (CompositeConnection connection = new
CompositeConnection("Host=myHost;Domain=myDomain;DataSource=myData
Source;User=myUser;Password=myPassword"))
  connection.Open()
End Using

Using (CompositeConnection connection = new
CompositeConnection("PoolIdleTimeout=-1;Host=myHost;Domain=myDomai
n;DataSource=myDataSource;User=myUser;Password=myPassword"))
//same connection pool
  connection.Open()
End Using

Using (CompositeConnection connection = new
CompositeConnection("Timeout=15;Host=myHost;Domain=myDomain;DataSo
urce=myDataSource;User=myUser;Password=myPassword")) //spawns a new
connection pool
  connection.Open()
End Using
```

Closing the Connection Pool

The connection pool is closed when the active process ends.

Calling Stored Procedures

You can invoke a stored procedure using CompositeCommand in the same way as any other SQL stored procedure. To instantiate a CompositeCommand object, provide the name of the stored procedure and a CompositeConnection instance as arguments to the constructor. Set the value of the CommandType property to "StoredProcedure" and add the parameters as key-value pairs to the Parameters collection of the CompositeCommand instance.

C#

```
string connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection connection = new
CompositeConnection(connectionString)) {
  CompositeCommand cmd = new CompositeCommand("SearchSuppliers",
connection);
  cmd.CommandType = CommandType.StoredProcedure;
  cmd.Parameters.Add(new CompositeParameter("@Country", "US"));
  // Add other parameters as needed ...

  CompositeDataReader rdr = cmd.ExecuteReader();
  while (rdr.Read()) {
    for (int i = 0; i < rdr.FieldCount; i++) {
      Console.WriteLine(rdr.GetName(i) + " --> " + rdr[i]);
    }
    Console.WriteLine();
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using connection As New CompositeConnection(connectionString)
  Dim cmd As New CompositeCommand("SearchSuppliers", connection)
  cmd.CommandType = CommandType.StoredProcedure
  cmd.Parameters.Add(New CompositeParameter("@Country", "US"))
  ' Add other parameters as needed ...

  Dim rdr As CompositeDataReader = cmd.ExecuteReader()
  While rdr.Read()
      For i As Integer = 0 To rdr.FieldCount - 1
          Console.WriteLine(rdr.GetName(i) + " --> " + rdr(i))
      Next
      Console.WriteLine()
  End While
End Using
```

Alternatively, you can set the parameters of a stored procedure in the text of the command. The support for stored procedure statements follows the standard form shown below:

```
"EXECUTE my_proc @first = 1, @second = 2, @third = 3;"

"EXEC my_proc @first = 1, @second = 2, @third = 3;"
```

To execute a parameterized query, add parameters as key-value pairs to the Parameters collection of the CompositeCommand instance.

C#

```
string connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection connection = new
CompositeConnection(connectionString)) {
  CompositeCommand cmd = new CompositeCommand("EXECUTE
SearchSuppliers Country = @Country;", connection);
  cmd.Parameters.Add(new CompositeParameter("@Country", "US"));
  // Add other parameters as needed ...
  CompositeDataReader rdr = cmd.ExecuteReader();
  while (rdr.Read()) {
    for (int i = 0; i < rdr.FieldCount; i++) {
      Console.WriteLine(rdr.GetName(i) + " --> " + rdr[i]);
    }
    Console.WriteLine();
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using connection As New CompositeConnection(connectionString)
  Dim cmd As New CompositeCommand("EXECUTE SearchSuppliers Country
= @Country;", connection)
  cmd.Parameters.Add(New CompositeParameter("@Country", "US"))
  ' Add other parameters as needed ...

  Dim rdr As CompositeDataReader = cmd.ExecuteReader()
  While rdr.Read()
      For i As Integer = 0 To rdr.FieldCount - 1
          Console.WriteLine(rdr.GetName(i) + " --> " + rdr(i))
      Next
      Console.WriteLine()
  End While
End Using
```

Using ADO.NET (Entity Framework)

Creating EF 6 Models from the Designer and Code

The ADO.NET Provider for TIBCO(R) Data Virtualization includes an Entity Framework 6 (EF6) provider: See Using EF 6 to set up a project. The following sections show how to build an EF data model that surfaces access to TDV tables.

Model-First Approach shows how to generate a model using schema introspection.

Code-First Approach shows how to manually define the entity definitions.

Using EF 6

The Entity Framework is now being developed outside of the .NET Framework. This means several dependencies that were part of the .NET framework are now part of the Entity Framework. This section describes how to use Entity Framework 6 (EF6) in your project.

Install Entity Framework 6

First, install and configure the EF6 environment. Use the Package Manager Console in Visual Studio to install the latest version of Entity Framework. Run the the following command to download and install Entity Framework automatically:

```
Install-Package EntityFramework
```

Register the Entity Framework Provider

Complete the following steps to register the Entity Framework provider:

1. Add the following provider entry in the "providers" section of your App.config or Web.config. This section should already exist if the Entity Framework installation was successful.

```
<configuration>
...
<entityFramework>
  <providers>
    ...
    <provider invariantName="System.Data.CompositeClient"
type="System.Data.CompositeClient.CompositeProviderServices,
System.Data.CompositeClient.Entities.EF6" />
  </providers>
</entityFramework>
</configuration>
```

2. Add a reference to System.Data.CompositeClient.Entities.EF6.dll, located in the lib > 4.0 subfolder in the installation directory.
3. Build the project to complete the setup for using EF6.

Build the Project

It is important to build the project so that the referenced assemblies are copied locally to the build location. These assemblies are used by the Visual Studio Entity Data Model wizard.

Using Entity Framework

With the setup complete, you can either use the Entity Data Model wizard described in Model-First Approach or use the code-first approach described in Code-First Approach.

Model-First Approach

This section describes how to use the Entity Data Model wizard to create the .edmx file and execute queries.

Install and Set Up Entity Framework

Install Entity Framework or add references to the required assemblies for your chosen version of Entity Framework. The assemblies are located in the lib subfolder of the installation directory. See Using EF 6 for using Entity Framework 6 (EF6). See Installed Assemblies for more information about all assemblies shipped with the provider

Add a New Item To Your Project

In the Visual Studio Solution Explorer, right-click your project and click Add > New Item.

Create an ADO.NET Entity Data Model

In the resulting Add New Item dialog, click ADO.NET Entity Data Model and enter an appropriate title, such as "CompositeEntityDataModel.edmx".

Use the Entity Data Model Wizard

Complete the following steps to create the .edmx file with the Entity Data Model wizard:

- On the first page, select Generate from database.
- On the next page, select the data provider connection you want to use with your project or create a new connection.
- Select whether to include sensitive data (such as passwords) in the connection string. Note that the option to exclude sensitive data means password fields are not copied to the generated config file with the rest of the connection string properties.
- Select the box to save the entity connection settings in App.Config, and enter a name for the context class of the data connection; for example, "CompositeEntities".

After Visual Studio retrieves the necessary information from the live data source, the wizard presents a listing of database objects you can include in your project. Note that this step may take several minutes as Visual Studio retrieves table schemas from TDV.

Perform LINQ Commands in Your Code

You are now ready to start using LINQ in your code.

```
CompositeEntities context = new CompositeEntities();
var ProductsQuery = from Products in context.Products
                   orderby Products.ProductName
                   select Products;
```

Note: Be sure to declare "using System.Linq" in your file.

Code-First Approach

An alternative to introspecting the model from the provider is to handwrite your model classes. This is the code-first approach to Entity Framework, which gives you greater control over the exact data model you use in your application.

Install Entity Framework

Install Entity Framework or add references to the required assemblies for your chosen version of Entity Framework. See Using EF 6 for using Entity Framework 6. See Installed Assemblies for more information about all assemblies shipped with the provider.

Register the Provider

Add the connection string to App.Config or Web.config. The connectionStrings node is often located directly below the configSection node in the root configuration node.

```
<configuration>
  ...
  <connectionStrings>
    <add name="CompositeContext"
connectionString="Host=myHost;Domain=myDomain;DataSource=myDataSou
rce;User=myUser;Password=myPassword"
providerName="System.Data.CompositeClient" />
  </connectionStrings>
  ...
</configuration>
```

Create the Context Class

Start by creating the CompositeContext class. This is the base object that extends DbContext and exposes the DbSet properties that represent the tables in the data source. Next, override some of the default functionality of the DbContext class by overriding the OnModelCreating method. You can find a description of these properties in the code below:

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.ModelConfiguration.Conventions;

class CompositeContext : DbContext {
  public CompositeContext() { }

  public DbSet<Products> Products { set; get; }

  protected override void OnModelCreating(DbModelBuilder
modelBuilder)
  {
    // To remove the requests to the Migration History table
    Database.SetInitializer<CompositeContext>(null);
    // To remove the plural names

modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
   //For versions of EF before 6.0, uncomment the following line to
remove calls to EdmTable, a metadata table
    //modelBuilder.Conventions.Remove<IncludeMetadataConvention>();
  }
}
```

Create the Table Models

Finally, define a class for each table that was defined in the DbSet properties of the context class. The table classes should have a list of properties that correspond to each field of that table. A corresponding map class must be defined to configure attributes for each property in the table class.

```
using System.Data.Entity.ModelConfiguration;
using System.ComponentModel.DataAnnotations.Schema; //EF 6 and
later
//using System.ComponentModel.DataAnnotations //For versions of EF
before 6


[System.ComponentModel.DataAnnotations.Schema.Table("Products")]
public class Products  {
  [System.ComponentModel.DataAnnotations.Key]
  public System.String Id { get; set; }
  public System.String ProductName { get; set; }
}
```

Perform LINQ Commands in Your Code

You are now ready to start using LINQ in your code. Be sure to declare "using System.Linq" in your file.

```
CompositeContext ents = new CompositeContext();
var ProductsQuery = from Products in ents.Products
                   orderby Products.ProductName
                   select Products;
```

Using ADO.NET (Entity Framework Core)

Creating EF 6 Models from the Designer and Code

The ADO.NET Provider for TIBCO(R) Data Virtualization includes an Entity Framework (EF) Core provider: See EFCore Console Application or EFCore ASP.NET Application to set up your desired project. The following sections show how to build an EF data model that surfaces access to TDV tables.

Code-First Approach shows how to manually define the entity definitions.

Reverse Engineering (Scaffolding) shows how to generate a model using scaffolding.

EFCore Console Application

Install Entity Framework Core

From a .NET core console app, install and configure the Entity Framework Core environment. Use the Package Manager Console in Visual Studio to install the latest version of EF Core. To download and install EF Core automatically, first run the following command:

```
Install-Package Microsoft.EntityFrameworkCore -Version 2.2.3
```

Additionally, you'll need to install EF Core's Relational assembly, as shown in the following example::

```
Install-Package Microsoft.EntityFrameworkCore.Relational -Version
2.2.3
```

Register the Entity Framework Core Provider

Complete the following steps to register the Entity Framework Core provider::

1. Add a reference to System.Data.CompositeClient.dll, located in the lib -> netstandard2.0 subfolder in the installation directory.
2. Add a reference to TIBCO.EntityFrameworkCore.Composite.dll, located in the lib -> netstandard2.0 -> EFCORE21 subfolder in the installation directory.
3. Build the project to complete the setup for using EF Core.

Creating the Data Model

There are two approaches that can be taken in creating the context and entity classes for your application. With the Code-First Approach approach, you can fine-tune your model by writing the classes manually. Alternatively, you can make use of Reverse Engineering (Scaffolding) to generate these classes automatically from your TDV schema.

Perform LINQ Commands in Your Code

You are now ready to start using LINQ in your code. Be sure to declare "using System.Linq" in your file.

```
CompositeContext ents = new CompositeContext();
var ProductsQuery = from Products in ents.Products
                   orderby Products.ProductName
                   select Products;
```

EFCore ASP.NET Application

Register the Entity Framework Core Provider

Complete the following steps to register the Entity Framework Core provider:

1. Add a reference to System.Data.CompositeClient.dll, located in the lib -> netstandard2.0 subfolder in the installation directory.
2. Add a reference to TIBCO.EntityFrameworkCore.Composite.dll, located in the lib -> netstandard2.0 -> EFCORE21 subfolder in the installation directory.
3. Build the project to complete the setup for using EF Core.

Creating the Data Model

There are two approaches that can be taken in creating the context and entity classes for your application. With the Code-First Approach approach, you can fine-tune your model by writing the classes manually. Alternatively, you can make use of Reverse Engineering (Scaffolding) to generate these classes automatically from your TDV schema.

Registering the Context with Dependency Injection

In order for the MVC controller to make use of the CompositeContext, you'll need to register it with dependency injection. Add the following to the beginning of your Startup.cs:

```
using MySolutionName.Models;
using Microsoft.EntityFrameworkCore;
```

Next, find the ConfigureServices method in Startup.cs and add the following at the end:

```
var connection =
@"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;
Password=myPassword";
services.AddDbContext<CompositeContext>(options =>
options.UseComposite(connection));
```

Creating a Controller and Views

To create a controller and views for your web app, follow the procedure below:

1. Right-click the Controllers folder in the Solution Explorer and navigate to Add -> Controller...
2. Choose MVC Controller with views, using Entity Framework, and click Add.
3. Set the Model Class to the class corresponding to your table/view and set the Data context class to CompositeContext.
4. Click Add. Note the name of the controller.

Running your Application

Now that the controllers and views have been setup, you can launch your app using Debug -> Start Without Debugging. The app will then launch in your browser. You can find your data by navigating to <Base URL of App>/<Name of controller minus the 'Controller.cs' at the end>.

Code-First Approach

An alternative to introspecting the model from the provider is to handwrite your model classes. This is the code-first approach to Entity Framework, which gives you greater control over the exact data model you use in your application.

Create the Context Class

This is the base object that extends DbContext and exposes the DbSet properties that represent the tables in the data source. Override some of the default functionality of the DbContext class by overriding the OnConfiguring method.

```
using Microsoft.EntityFrameworkCore;

public class CompositeContext : DbContext
{
    public DbSet<Products> Products { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {

optionsBuilder.UseComposite("Host=myHost;Domain=myDomain;DataSourc
e=myDataSource;User=myUser;Password=myPassword");
        }
    }
}
```

Create the Table Models

Define a class for each table that was defined in the DbSet properties of the context class. The table classes should have a list of properties that correspond to each field of that table. A corresponding map class must be defined to configure attributes for each property in the table class.

```
public class Products
```

```
{
    public string Id { get; set; }
    public string ProductName { get; set; }
}
```

Reverse Engineering (Scaffolding)

Reverse engineering (via scaffolding) is a process which streamlines OR/M by automatically constructing classes for all of the tables and views available via the TDV schema. This process also creates a TIBCOContext class, which extends DbContext and exposes the DbSet properties that represent the tables in the data source.

Install Entity Framework Core Tools

If you're making use of scaffolding in a console app, you'll first need to install the EF Tools via the Package Manager Console (PMC).

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

Scaffolding

Scaffolding is performed using the PMC in Visual Studio. You can use following commands to scaffold.

Scaffold All Tables

Use the following command to scaffold all tables and views from the schema into your Models folder:

```
Scaffold-DbContext
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword" TIBCO.EntityFrameworkCore.Composite -OutputDir
Models -Context CompositeContext
```

Scaffold A Subset of Tables

You can also refine the scaffolding process to only create classes for a limited selection of tables. This is especially useful when a large schema is taking a long time to scaffold due to the large number of classes it has to generate. This is accomplished by specifying the tables/views that you want to scaffold at the end of your Scaffold-DbContext command in the PMC.

```
Scaffold-DbContext
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword" TIBCO.EntityFrameworkCore.Composite -OutputDir
Models -Context CompositeContext -Tables Products
```

Updating your Data Model

If you would like to add re-scaffold to update the model additional table classes after the initial generation, simply add a '-Force' argument to any Scaffold-DbContext command. The existing model will then be overwritten with your changes.

Using ADO.NET (LINQ)

LINQ to TDV

The following sections show how to query and manipulate the TDV objects surfaced in an Entity Framework data model.

LINQ Queries

LINQ Updates

LINQ Inserts

LINQ Deletes

Stored Procedures

LINQ Queries

Below are examples of LINQ queries supported by the provider.

Select and Filter

The following query searches for records that have a value of "Konbu" for the ProductName column.

```
var query = from Products in context.Products
            where Products.ProductName == "Konbu"
            select Products;
```

Contains

You can also search for strings within strings: The following example retrieves all Products entities with a value that contains "B" in the ProductName column.

```
var query = from Products in context.Products
            where Products.ProductName.Contains("B")
            select Products;
```

Limit

The following example limits the results returned to 10.

```
var query = (from Products in context.Products
            orderby Products.ProductName descending,
            Products.ProductName ascending
            select Products).Take(10);
```

Order By

The following example executes a multicolumn sort: Results are sorted by Id in ascending order (the default), then by ProductName in descending order.

```
var query = from Products in context.Products;
            orderby Products.Id,
                    Products.ProductName descending
            select Products;
```

Count

The following example counts all entities that match a given criterion.

```
int count = (from Products in context.Products
             where Products.ProductName == "Konbu"
             select Products).Count();
```

Group and Summarize

The following example calculates the minimum Price for a collection of entities grouped by ProductName, the key defined in the group clause.

```
var query = from Products in context.Products
            group Products by new { Products.ProductName }
            into g
            orderby g.Key
            select new {
              ProductName = g.Key.ProductName,
              MinPrice = g.Min(x => x.Price)
            };
```

Joins

The following is an example of an inner join. The following query returns the Names of all Accounts that have Contacts and the Names of those Contacts:

```
var AccountsQuery = from accounts in context.Accounts
                  join contacts in context.Contacts on accounts.Id
equals contacts.AccountId
                    select new
                    {
                      AccountName = accounts.Name,
                      ContactName =  contacts.Name
                    };
```

The following is an example of a left outer join. The query returns the Names for all Accounts along with the Names of any associated Contacts:

```
var AccountsQuery = from accounts in context.Accounts
                  join contacts in context.Contacts on accounts.Id
equals contacts.AccountId into Inners
                    from od in Inners.DefaultIfEmpty()
                    select new
                    {
                      AccountName = accounts.Name,
                      ContactName =  od.ContactName
                    };
```

The following is an example of a cross join. The query combines every row from the Employees table with every row from the sales territories table and then projects the FirstName, LastName, and TerritoryDescription columns over these rows:

```
var infoQuery =
                from emp in context.Employees
                from empterr in context.EmployeeTerritories
                select new
                {
                    emp.FirstName,
                    emp.LastName,
                    empterr.TerritoryDescription
                };
```

LINQ Updates

The following sections show several ways to update TDV objects with LINQ.

Updating TDV Objects with LINQ to Entities

The following example uses LINQ query syntax to obtain the object to be updated, then updates the object's properties. Note that updates only work on one record at a time, selected by its Id in the WHERE clause.

```
using System.Linq;

CompositeEntities context = new CompositeEntities();

var ProductsQuery = from Products in context.Products
                   where Products.Id == "22"
                   select Products;

foreach (var result in ProductsQuery) {
  result.ProductName = "Konbu";
}

try {
  context.SaveChanges();
} catch (Exception e) {
  Console.WriteLine(e);
}
```

Using LINQ Element Operators

The SingleOrDefault and FirstOrDefault element operators provide alternative ways to obtain an object you want to manipulate. You can also use these element operators to simplify updating multiple records.

Selecting a Single Entity to Update

Another way to accomplish the same result as the preceding LINQ query is to use the SingleOrDefault element operator, as shown in the following example:

```
CompositeEntities context = new CompositeEntities();
var Products = context.Products.SingleOrDefault(o => o.Id ==
"150000-933272658");
Products.ProductName = "Ikura";
context.Entry(Products).State =
System.Data.Entity.EntityState.Modified;
context.SaveChanges();
```

Updating a Set of Entities

You can update a set of entities by performing multiple queries. The next example builds on the previous example to update an entire set of records and work around the limitation imposed by LINQ of one record per update. Our initial query retrieves a set of records that match the condition (in this instance, all the records whose ProductName is "Konbu").

Separate queries are then performed for each Id, whose descriptions are then changed and saved. Note the performance penalty for this approach; a separate connection must be established for each desired update.

```
CompositeEntities context = new CompositeEntities();

//Select everything matching the condition
var ProductsQuery = from Products in context.Products
                   where Products.ProductName == "Konbu"
                   select Products;

foreach (var result in ProductsQuery) {
  //For each record matching the condition, perform a separate
  //command to update the attributes you desire to change.
  var updateRecord = from Products in context.Products
                     where Products.Id == result.Id
                     select Products;

  //Since the record was selected using the record's unique primary
key, you can derive the updateRecord using the
  //FirstOrDefault method.
  updateRecord.FirstOrDefault().Description = "test desc";

  try {
    //Commit the changes to the database.
    context.SaveChanges();
  } catch (Exception e) {
    Console.WriteLine(e);
  }
}
```

LINQ Inserts

This section provides an example of how to obtain and insert TDV objects with LINQ.

Inserting TDV Objects with LINQ to Entities

The following code adds a new Products object:

```
using System.Linq;

CompositeEntities context = new CompositeEntities();

Products newRecord = new Products {
  ProductName = "Konbu"
};

context.Products.Add(newRecord);

try {
  context.SaveChanges();
  Console.WriteLine(newRecord.Id);
} catch (Exception e) {
  Console.WriteLine(e);
}
```

Note: It is often useful to retrieve the result of the insert command. For example, when obtaining the Id of the new record. You can do so by reading the Id from the record after it has been saved. The primary key, generated by the data source, is assigned to the saved record.

LINQ Deletes

This section describes several ways to delete TDV objects with LINQ.

Deleting TDV Objects with LINQ to Entities

The following example uses LINQ query syntax to obtain the object to be deleted, then deletes the object from the data model. Note that deletes only work on one record at a time, selected by its Id in the WHERE clause.

```
CompositeEntities context = new CompositeEntities();

var query = from Products in context.Products
            where Products.Id == "22"
            select Products;

context.Products.Remove(query.FirstOrDefault());

try {
  context.SaveChanges();
} catch (Exception e) {
  Console.Write(e.Message);
}
```

Using LINQ Element Operators

The SingleOrDefault and FirstOrDefault element operators provide alternative ways to obtain an object you want to delete. You can also use these element operators to simplify deleting multiple records.

Selecting a Single Entity to Delete

Another way to accomplish the same result as the preceding query is to use the SingleOrDefault element operator. For example:

```
CompositeEntities context = new CompositeEntities();
var Products = context.Products.SingleOrDefault(o => o.Id ==
"150000-933272658");
context.Entry(Products).State =
System.Data.Entity.EntityState.Deleted;
context.SaveChanges();
```

Deleting a Set of Entities

The following example deletes a group of records whose ProductName value equals "Konbu". Note that LINQ only performs one delete operation per record at a time as specified by the Id of the record. Because of this, LINQ incurs a performance penalty as it opens a fresh connection with the data source every time it deletes a record.

```
CompositeEntities context = new CompositeEntities();

var query = from Products in context.Products
            where Products.ProductName == "Konbu"
            select Products;

foreach (var result in query) {
  var delete = from Products in context.Products
               where Products.Id == result.Id
               select Products;
  context.Products.Remove(delete.FirstOrDefault());
}

try {
  context.SaveChanges();
} catch (Exception e) {
  Console.Write(e.Message);
}
```

Stored Procedures

The following example shows how to call a stored procedure using Entity Framework.

Calling Stored Procedures with SQL

To execute a stored procedure, you execute SQL directly through the Entity Framework context: Define a class to capture the results of the stored procedure; for example, the Name class shown below. This class should define a property for each output column that you are interested in using. Once this class is defined, you can invoke the ExecuteStoreQuery method of the Entity Framework context.

C#

```
public class SearchSuppliersOutput {
  public string Name { get; set; }
  // ... add other return values with matching data type
}

public class Program {
  static void Main(string[] args) {
    using (CompositeEntities context = new CompositeEntities())
    {
      //use the line below for versions earlier than EF 5:
      //IEnumerable<SearchSuppliersOutput> result =
context.ExecuteStoreQuery<SearchSuppliersOutput>("EXEC
SearchSuppliers @Country = ?;",new CompositeParameter("Country",
"US"));
      //use the line below for EF 5 and 6:
      IEnumerable<SearchSuppliersOutput> result =
context.Database.SqlQuery<SearchSuppliersOutput>("EXEC
SearchSuppliers @Country = ?", new CompositeParameter("Country",
"US"));
      List<SearchSuppliersOutput> resultList =
result.ToList<SearchSuppliersOutput>();
      foreach (SearchSuppliersOutput value in resultList) {
        Console.WriteLine("Name: " + value.Name);
      }
    }
  }
}
```

VB.NET

```
Public Class SearchSuppliersOutput
    Public Property Name() As String
  ' ... add other return values with matching data type
End Class

Module Program
  Sub Main
    Using context As New CompositeEntities
      'use the line below for versions earlier than EF 5:
      'Dim result As IEnumerable(Of SearchSuppliersOutput) =
context.ExecuteStoreQuery(Of SearchSuppliersOutput)("EXEC
SearchSuppliers @Country = ?;", New CompositeParameter("Country",
"US"))
      'use the line below for EF 5 and 6:
      Dim result As IEnumberable(Of SearchSuppliersOutput) =
context.Database.SqlQuery(Of SearchSuppliersOutput)("EXEC
SearchSuppliers @Country = ?;", New CompositeParameter("Country",
"US"))
      Dim resultList As List(Of SearchSuppliersOutput) =
result.ToList()
      For Each value As SearchSuppliersOutput In resultList
        Console.WriteLine("Name: " + value.Name)
      Next
    End Using
  End Sub
End Module
You can avoid using CompositeParameter by passing in the parameters
directly, as shown in the following example:

"EXEC SearchSuppliers @Country=US, ..."
```

Using ADO.NET (SSRS)

You can use the ADO.NET Provider for TIBCO(R) Data Virtualization to integrate real-time connectivity to TDV into your SSRS reports. The provider supports SSRS versions 2005 and above.

Create and Publish Reports in the Report Designer

Complete the following steps to publish reports to servers running in native mode or in SharePoint mode, using the Report Designer tool in Visual Studio:

Deploy the Provider

Create a Data Source

Create a Dataset

Publish a Report

Deploy the Provider

The ADO.NET Provider for TIBCO(R) Data Virtualization automates SSRS deployment, adding the report type "TIBCO TDV Report." This section shows the typical deployment flow. See Installed Assemblies for more information on the SSRS assemblies used by the provider.

Deploy to Native-Mode Servers

The provider installation provides an option to automatically deploy the provider on native mode report servers.

Sharepoint-Mode Report Servers

On SharePoint mode report servers, you can run the install-sprs.ps1 PowerShell script. Run the script from the lib subfolder in the installation directory.

Create a Data Source

Follow the steps below to create the data source, providing authentication and any necessary connection properties.

Share or Embed the Data Source

You can create shared data sources or create an embedded data source restricted to a single port.

1. Open your report or create a new Report Server project and add a new report by right-clicking the Reports folder in Solution Explorer and clicking Add -> New Item -> Report.
2. If you want to create a shared data source, right-click Shared Data Sources in Solution Explorer and click Add New Data Source. If you want to create an embedded data source, right-click Data Sources in the Report Data view.
3. Enter a name for the data source and in the Type menu select TIBCO Composite Report.
4. In the Connection String box, enter the connection string to connect to TDV. A typical connection string is below:
   ```
   Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser
   ;Password=myPassword
   ```

See Establishing a Connection for a connection and authentication how-to.

Create a Dataset

Follow the steps below to use Visual Studio wizards to define the query that will provide the report's dataset.

Share or Embed TDV Datasets

Like SSRS data sources, datasets can be shared with other reports or embedded in a single report. Both types access the remote data without making a copy; report definitions in SSRS do not contain the actual data.

1. If you want to create a shared dataset, right-click Shared Datasets in Solution Explorer and click Add New Dataset. If you want to create an embedded dataset, right-click Datasets in the Report Data view, click Add Dataset, and select the option to use a dataset embedded in your report.
2. Select the data source you created.
3. Click Query Designer to build the query visually and preview the results. Or, enter a query in the Query box:
   ```
   SELECT * FROM Products WHERE ProductName = 'Konbu'
   ```
4. To create a parameterized query, use the following syntax and click Run in the Query Designer.
   ```
   SELECT * FROM Products WHERE ProductName = @ProductName
   ```

   When you create the dataset, a report parameter is associated with the dataset parameter you defined.

Publish a Report

Follow the steps below to design a simple report using Tablix and publish the report from Visual Studio.

Design the Report

Use Tablix (table, list, and matrix) data regions as the base for the design of your report: Select a data region, such as a Table, in the Toolbox, and then click the design surface. The table is added to the designer. You can then drag columns from the dataset to columns in the table.

Preview and Publish the Report

To retrieve live data into the report, click the Preview tab in the Designer. For a parameterized query, specify a parameter and then click View Report.

To publish a report to a report server or to a SharePoint site, configure the report folder and the URL of the report server in the project properties:

- If you are publishing to SharePoint, the values for all properties must be fully qualified URLs.
- For example, a native mode report server has the following syntax: http://MyServerName/ReportServer. A SharePoint mode report server has the following syntax: http://MyServerName/MySite/MySubsite.
- After you have defined the project configuration, you can deploy the report project or a single report by right-clicking the object in Solution Explorer and clicking Deploy.

Using ADO.NET (DbProviderFactory)

The ADO.NET Provider for TIBCO(R) Data Virtualization implements the CompositeProviderFactory class to enable you to write generic data access code to TDV through the ADO.NET base classes.

Creating Data Access Objects with CompositeProviderFactory

The following sections show how to use the CompositeProviderFactory class to create objects like CompositeConnection, CompositeCommand, and CompositeDataAdapter in a generic way.

Creating DbConnections

Executing DbCommands

Creating DbConnections

You can use DbProviderFactory and DbConnection objects to connect to TDV with generic code. This section describes how to connect from your project.

Adding Provider Information to the Configuration Context

To create a strongly typed DbProviderFactory, you must first register the ADO.NET Provider for TIBCO(R) Data Virtualization in the configuration context (machine.config, app.config, or web.config). Note that the provider installer registers the provider in machine.config.

If you are not using the installer -- for example, if you are using the NuGet package instead -- you need to manually register the provider. You can do so by adding an entry to the System.Data section of the configuration context. You can modify the System.Data section in your machine.config or app.config (the System.Data in the app config is merged with machine.config at run time). Below is the syntax and format of the configuration entry:

```
<system.data>
  <DbProviderFactories>
    <add name="ADO.NET Provider for TIBCO(R) Data Virtualization"
invariant="System.Data.CompositeClient" description="ADO.NET
Provider for TIBCO(R) Data Virtualization"
type="System.Data.CompositeClient.CompositeProviderFactory,
System.Data.CompositeClient, Version=19.0.0.40, Culture=neutral,
PublicKeyToken=cdc168f89cffe9cf" />
</DbProviderFactories>
</system.data>
```

The following configuration file fragment defines a typical TDV connection string in the context:

```
<configuration>
  <connectionStrings>
    <add name="TDV"
      providerName="System.Data.CompositeClient"

connectionString="Host=myHost;Domain=myDomain;DataSource=myDataSou
rce;User=myUser;Password=myPassword"
    />
  </connectionStrings>
</configuration>
```

Creating the DbProviderFactory

Call **DbProviderFactories.GetFactory** to create the DbProviderFactory:

```
DbProviderFactory factory =
DbProviderFactories.GetFactory("System.Data.CompositeClient");
```

DbProviderFactories looks up the invariant name in the configuration context to find the assembly and the CompositeProviderFactory class.

Creating the DbProviderFactory and DbConnection

The following code shows how to create a strongly typed DbProviderFactory object and use it to instantiate a DbConnection object and connect to TDV.

```
DbProviderFactory factory =
DbProviderFactories.GetFactory("System.Data.CompositeClient");

using(DbConnection connection = factory.CreateConnection()) {
  connection.ConnectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";
  connection.Open();
}
```

You can also read the connection string from the application configuration file. Note that the ConnectionStringSettingsCollection class requires a reference to System.Configuration.dll.

The following code retrieves the first TDV connection string defined in the application configuration file.

```
ConnectionStringSettingsCollection settings =
ConfigurationManager.ConnectionStrings;

if (settings != null) {
  foreach (ConnectionStringSettings cs in settings) {
    if (cs.ProviderName == "System.Data.CompositeClient")
      connection = cs.ConnectionString;
    break;
  }
}
```

Executing DbCommands

The following section shows how to execute commands using ADO.NET base classes like DbCommand and DbDataAdapter.

Executing Commands to TDV

The following code executes a "SELECT *" query to TDV, given an existing DbConnection object.

```
using (connection) {

  // Create the DbCommand.
  DbCommand command = factory.CreateCommand();
  command.CommandText =
    "SELECT * FROM Products";

  command.Connection = connection;

  // Create the DbDataAdapter.
  DbDataAdapter adapter = factory.CreateDataAdapter();
  adapter.SelectCommand = command;

  // Fill the DataTable.
  DataTable table = new DataTable();
  adapter.Fill(table);

  //  Display each row and column value.
  foreach (DataRow row in table.Rows) {
    foreach (DataColumn column in table.Columns) {
      Console.WriteLine(row[column]);
    }
  }
}
```

Schema Discovery

The provider supports schema discovery using ADO.NET classes or using SQL queries to the available system tables. The ADO.NET classes enable access to schema information, connection property information, and information on the columns returned.

Through SQL queries to the available System Tables, you can access schema and connection property information as well as information on data source functionality and statistics on update operations.

Using ADO.NET Schema Collections

You can use ADO.NET schema collections to retrieve schema and connection property information. Invoke the GetSchema method of the CompositeConnection class to access the following metadata:

Tables

Views

Columns

Procedures

Procedure Parameters

Indexes

Index Columns

Foreign Keys

Databases

Users

Connection Properties

Using Result Set Column Information

To access information about the columns returned by a query, invoke the GetSchemaTable method of the CompositeDataReader class. See Result Sets for code examples.

Using SQL

You can query the System Tables to access any metadata surfaced through the provider.

Tables

The Tables schema collection lists all tables in the database, including views.

Retrieving the Table Listing

To retrieve the Tables schema collection, call the GetSchema method of the CompositeConnection class.

C#

```
String connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection conn = new
CompositeConnection(connectionString)) {
  conn.Open();
  DataTable databaseSchema = conn.GetSchema("Tables");
  foreach (DataRow row in databaseSchema.Rows) {
    Console.WriteLine(row["TABLE_NAME"]);
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using conn As New CompositeConnection(connectionString)
  conn.Open()
  Dim databaseSchema As DataTable = conn.GetSchema("Tables")
  For Each row As DataRow In databaseSchema.Rows
    Console.WriteLine(row("TABLE_NAME"))
  Next
End Using
```

Columns Returned

The Tables schema collection returns the following columns.

Column Name	Data Type	Description
TABLE_CATALOG	System.String	The database that contains the table.
TABLE_SCHEMA	System.String	The schema that contains the table.
TABLE_NAME	System.String	The table name.
TABLE_TYPE	System.String	The table type.

Views

ou can retrieve the listing of views from the Views schema collection.

Retrieving the View Listing

To retrieve the Views schema collection, call the GetSchema method of the CompositeConnection class, as shown in the following examples.

C#

```
String connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection conn = new
CompositeConnection(connectionString)) {
  conn.Open();
  DataTable databaseSchema = conn.GetSchema("Views");
  foreach (DataRow row in databaseSchema.Rows) {
    Console.WriteLine(row["TABLE_NAME"]);
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using conn As New CompositeConnection(connectionString)
  conn.Open()
  Dim databaseSchema As DataTable = conn.GetSchema("Views")
  For Each row As DataRow In databaseSchema.Rows
    Console.WriteLine(row("TABLE_NAME"))
  Next
End Using
```

Columns Returned

The Views schema collection returns the following columns:

Column Name	Data Type	Description
TABLE_CATALOG	System.String	The database that contains the view.
TABLE_SCHEMA	System.String	The schema that contains the view.
TABLE_NAME	System.String	The view name.
CHECK_OPTION	System.String	Whether the view was created using WITH CHECK OPTION.
IS_UPDATEABLE	System.String	Whether the view is updateable.

Columns

To access metadata for the columns available in the database, retrieve the Columns schema collection. You can also access the column metadata for views by retrieving the ViewColumns schema collection.

Alternatively, retrieve metadata from Result Sets. The same columns are returned for result set metadata as the Columns and ViewColumns schema collections; see below.

Retrieving Column Metadata

Call the GetSchema method of the CompositeConnection class to retrieve the Columns or ViewColumns schema collections. You can restrict results by table name, as shown in the example below.

C#

```
string connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";
```

```
using (CompositeConnection conn = new
CompositeConnection(connectionString)) {
  conn.Open();
  DataTable databaseSchema = conn.GetSchema("Columns", new string[]
{"Products"});
  foreach (DataRow column in databaseSchema.Rows) {
    Console.WriteLine(column["COLUMN_NAME"]);
    Console.WriteLine(column["IS_KEY"]);
    Console.WriteLine(column["DATA_TYPE"]);
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using conn As New CompositeConnection(connectionString)
  conn.Open()
  Dim databaseSchema As DataTable = conn.GetSchema("Columns", New
String() {"Products"})
  For Each column As DataRow In databaseSchema.Rows
    Console.WriteLine(column("COLUMN_NAME"))
    Console.WriteLine(column("IS_KEY"))
    Console.WriteLine(column("DATA_TYPE"))
  Next
End Using
```

Columns Returned

The Columns and ViewColumns schema collections and DataTables returned from the query contain the following columns.

Column Name	Data Type	Description
TABLE_CATALOG	System.String	The database containing the table.
TABLE_SCHEMA	System.String	The schema containing the table.
TABLE_NAME	System.String	The table containing the column.
COLUMN_NAME	System.String	The column name.

Column Name	Data Type	Description
ORDINAL_POSITION	System.Int32	The sequence number of the column.
COLUMN_DEFAULT	System.String	The default value of the column.
IS_NULLABLE	System.String	Whether the column allows null values. This value is YES or NO.
DATA_TYPE	System.String	The column data type.
CHARACTER_MAXIMUM_LENGTH	System.Int32	The maximum length in characters of a column with character data.
NUMERIC_PRECISION	System.Int32	The maximum number of digits allowed for numeric data.
NUMERIC_SCALE	System.Int32	The maximum column scale or the number of digits to the right of the decimal point in numeric data.
DATETIME_PRECISION	System.Int32	Returns the precision in fractional seconds if the parameter type is datetime or smalldatetime. Otherwise, returns NULL.
CHARACTER_SET_NAME	System.String	The name of the character set for a column with character data.
COLUMN_COMMENT	System.String	A brief description of the column.
IS_KEY	System.Boolean	Whether the column is the primary key of the table referenced by TABLE_NAME.
IS_READONLY	System.Boolean	Whether the column is read-only.
PROVIDER_TYPE	System.Type	Indicates the appropriate data type dependent on the language you are executing in.

Procedures

The Procedures schema collection describes the available stored procedures.

Retrieving the Stored Procedure Listing

To retrieve the Procedures schema collection, call the GetSchema method of the CompositeConnection class. Access the metadata in the DataTable object returned.

The following example outputs a list of stored procedure names:

C#

```
String connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection conn = new
CompositeConnection(connectionString)) {
  conn.Open();
  DataTable table = conn.GetSchema("Procedures");
  foreach (DataRow row in table.Rows)
   Console.WriteLine(row["SPECIFIC_NAME"]);
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using conn As New CompositeConnection(connectionString)
  conn.Open()
  Dim table As DataTable = conn.GetSchema("Procedures")
  For Each row As DataRow in table.Rows
    Console.WriteLine(row("SPECIFIC_NAME"))
  Next
End Using
```

Columns Returned

The Procedures schema collection contains the following columns:

Column Name	Data Type	Description
SPECIFIC_CATALOG	System.String	The name of the database containing the stored procedure.
SPECIFIC_SCHEMA	System.String	The schema that contains the stored procedure.
SPECIFIC_NAME	System.String	The name of the stored procedure containing the parameter.
ROUTINE_CATALOG	System.String	The database containing the stored procedure.
ROUTINE_SCHEMA	System.String	The schema containing the stored procedure.
ROUTINE_NAME	System.String	The name of the stored procedure.
ROUTINE_TYPE	System.String	Returns PROCEDURE for stored procedures and FUNCTION for functions.

Procedure Parameters

The ProcedureParameters schema collection describes the stored procedure parameters.

Retrieving Stored Procedure Parameter Metadata

The ProcedureParameters schema collection contains information about the parameters of stored procedures.

To retrieve the ProcedureParameters schema collection, call the GetSchema method of the CompositeConnection class. Access the metadata in the DataTable object returned. The following example retrieves parameter information for all stored procedures:

C#

```
String connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection conn = new
CompositeConnection(connectionString)) {
  conn.Open();
  DataTable table = conn.GetSchema("ProcedureParameters");
  foreach (DataRow row in table.Rows) {
    foreach (DataColumn col in table.Columns) {
      Console.WriteLine(col.ColumnName + "=" + row[col]);
    }
  }
}
```

VB.NET

```
Dim connectionString As [String] =
"User=Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myU
ser;Password=myPassword"

Using conn As New CompositeConnection(connectionString)
  conn.Open()
  Dim table As DataTable = conn.GetSchema("ProcedureParameters")
  For Each row As DataRow In table.Rows
    For Each col As DataColumn In table.Columns
      Console.WriteLine(col.ColumnName + "=" + row(col))
    Next
  Next
End Using
```

Columns Returned

The columns of the schema collection are the following:

Column Name	Data Type	Description
SPECIFIC_CATALOG	System.String	The name of the database containing the stored procedure.
SPECIFIC_SCHEMA	System.String	The schema that contains the stored procedure.
SPECIFIC_NAME	System.String	The name of the stored procedure containing the parameter.

Column Name	Data Type	Description
PARAMETER_NAME	System.String	The name of the parameter.
PARAMETER_MODE	System.String	Returns IN for an input parameter, OUT for an output parameter, or INOUT for parameters that can be both input and output parameters.
ORDINAL_POSITION	System.Int32	The sequence number of the parameter.
DATA_TYPE	System.String	The data type name.
CHARACTER_MAXIMUM_LENGTH	System.Int32	The maximum length in characters.
CHARACTER_SET_NAME	System.String	The name of the character set for a column with character data.
NUMERIC_PRECISION	System.Int32	The maximum number of digits in numeric data.
NUMERIC_SCALE	System.Int32	The column scale or number of digits to the right of the decimal point.
DATETIME_PRECISION	System.Int32	The precision in fractional seconds if the parameter type is datetime or smalldatetime. Otherwise, returns NULL.
PROCEDURE_DESCRIPTION	System.String	A brief description of the procedure.
PROVIDER_TYPE	System.Type	Indicates the appropriate data type dependent on the language you are executing in.

Indexes

You can retrieve information on indexes, such as the primary keys, by querying the Indexes collection.

Retrieving Primary Key Information

To retrieve this schema collection, call the GetSchema method of the CompositeConnection class. You can restrict the results by table name. The following example retrieves the primary key of the TDV table Products.

C#

```
String connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection conn = new
CompositeConnection(connectionString)) {
  conn.Open();
  DataTable databaseSchema = conn.GetSchema("Indexes", new string[]
{"Products"});
  foreach (DataRow row in databaseSchema.Rows) {
    Console.WriteLine(row["INDEX_NAME"]);
    Console.WriteLine(row["PRIMARY"]);
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using conn As New CompositeConnection(connectionString)
  conn.Open()
  Dim databaseSchema As DataTable = conn.GetSchema("Indexes",  New
String() {"Products"})
  For Each row As DataRow In databaseSchema.Rows
    Console.WriteLine(row("INDEX_NAME"))
    Console.WriteLine(row("PRIMARY"))
  Next
End Using
```

Columns Returned

The Indexes schema collection contains the following columns:

Column Name	Data Type	Description
INDEX_CATALOG	System.String	The name of the database containing the index.
INDEX_SCHEMA	System.String	The name of the schema containing the index.
TABLE_NAME	System.String	The name of the table containing the index.
INDEX_NAME	System.String	The name of the index.
UNIQUE	System.Boolean	Whether the index is unique.
PRIMARY	System.Boolean	Whether the index is a primary key.
TYPE	System.Int32	An integer value corresponding to the index type: statistic (0), clustered (1), hashed (2), or other (3).
COMMENT	System.String	A description of the index.

Index Columns

The IndexColumns schema collection lists the indexes and their corresponding columns. By filtering on indexes, you can write more selective queries with faster query response times.

Retrieving Index Column Information

To retrieve this schema collection, call the GetSchema method of the CompositeConnection class. You can restrict the results by table name. The following example retrieves the column and sequence number for each index of the TDV table Products.

C#

```
String connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection conn = new
CompositeConnection(connectionString)) {
  conn.Open();
  DataTable databaseSchema = conn.GetSchema("IndexColumns", new
string[] {"Products"});
  foreach (DataRow row in databaseSchema.Rows) {
    Console.WriteLine(row["COLUMN_NAME"]);
    Console.WriteLine(row["ORDINAL_POSITION"]);
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using conn As New CompositeConnection(connectionString)
  conn.Open()
  Dim databaseSchema As DataTable = conn.GetSchema("IndexColumns",
New String() {"Products"})
  For Each row As DataRow In databaseSchema.Rows
    Console.WriteLine(row["COLUMN_NAME"])
    Console.WriteLine(row["ORDINAL_POSITION"])
  Next
End Using
```

Columns Returned

The IndexColumns schema collection returns the following columns:

Column Name	Data Type	Description
INDEX_CATALOG	System.String	The name of the database containing the index.
INDEX_SCHEMA	System.String	The name of the schema containing the index.
TABLE_NAME	System.String	The name of the table containing the index.
INDEX_NAME	System.String	The name of the index.

Column Name	Data Type	Description
COLUMN_NAME	System.String	The name of the column associated with the index.
ORDINAL_POSITION	System.Int32	The sequence number of the column.
SORT_ORDER	System.Int32	Returns A for ascending and D for descending.

Foreign Keys

This section describes how to access information about foreign keys by retrieving the ForeignKeys schema collection.

Retrieving Foreign Key Information

To retrieve the ForeignKeys schema collection, call the GetSchema method of the CompositeConnection class. You can restrict foreign key information by the table name.

Access the results in the DataTable returned. The following example lists the foreign keys for the Products table.

C#

```
String connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection conn = new
CompositeConnection(connectionString)) {
  conn.Open();
  DataTable databaseSchema = conn.GetSchema("ForeignKeys", new
string[] {"Products"});
  foreach (DataRow row in databaseSchema.Rows) {
    Console.WriteLine(row["CONSTRAINT_NAME"]);
    Console.WriteLine(row["TABLE_NAME"]);
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using conn As New CompositeConnection(connectionString)
  conn.Open()
  Dim databaseSchema As DataTable = conn.GetSchema("ForeignKeys",
New String() {"Products"})
    For Each row As DataRow In databaseSchema.Rows
    Console.WriteLine(row("CONSTRAINT_NAME"))
    Console.WriteLine(row("TABLE_NAME"))
  Next

End Using
```

Columns Returned

The ForeignKeys schema collection returns the following information about the foreign keys in TDV.

Column Name	Data Type	Description
CONSTRAINT_CATALOG	System.String	The database containing the foreign key.
CONSTRAINT_SCHEMA	System.String	The schema containing the foreign key.
CONSTRAINT_NAME	System.String	The name of the foreign key.
CONSTRAINT_TYPE	System.String	Returns FOREIGN KEY.
TABLE_CATALOG	System.String	The database of the table containing the foreign key.
TABLE_SCHEMA	System.String	The schema of the table containing the foreign key.
TABLE_NAME	System.String	The name of the table containing the foreign key.
IS_DEFERRABLE	System.String	Whether the foreign key is deferrable. This value is YES or NO.

Column Name	Data Type	Description
INITIALLY_DEFERRED	System.String	Whether the foreign is initially deferrable. This value is YES or NO.

Databases

The Databases schema collection lists all the available databases.

Retrieving the Database Listing

To retrieve the Databases schema collection, call the GetSchema method of the CompositeConnection class.

C#

```
String connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection connection = new
CompositeConnection(connectionString)) {
  connection.Open();
  DataTable databaseSchema = connection.GetSchema("Databases");
  foreach (DataRow row in databaseSchema.Rows) {
    Console.WriteLine(row["Database"]);
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using connection As New CompositeConnection(connectionString)
  connection.Open()
  Dim databaseSchema As DataTable =
connection.GetSchema("Databases")
  For Each row As DataRow In databaseSchema.Rows
    Console.WriteLine(row("Database"))
  Next
```

```
End Using
```

Columns Returned

The Databases schema collection returns the following columns.

Column Name	Data Type	Description
Database	System.String	The database name.
Type	System.String	The database type.
Owner	System.String	The owner of the database.

Users

The Users schema collection lists all users in the database.

Retrieving the Users Listing

To retrieve the Users schema collection, call the GetSchema method of the CompositeConnection class.

C#

```
String connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection connection = new
CompositeConnection(connectionString)) {
  connection.Open();
  DataTable databaseSchema = connection.GetSchema("Users");
  foreach (DataRow row in databaseSchema.Rows) {
    Console.WriteLine(row["User"]);
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"

Using connection As New CompositeConnection(connectionString)
  connection.Open()
  Dim databaseSchema As DataTable = connection.GetSchema("Users")
  For Each row As DataRow In databaseSchema.Rows
    Console.WriteLine(row("User"))
  Next
End Using
```

Columns Returned

The Users schema collection returns the following columns.

Column Name	Data Type	Description
User	System.String	The user name.
Domain	System.String	The domain of the user.
Annotation	System.String	The annotations for the user.

Connection Properties

You can programmatically access the information about the available connection properties and those set in the connection string by querying the ConnectionProperties schema collection.

Retrieving Connection Property Information

To retrieve the ConnectionProperties schema collection, call the GetSchema method of the CompositeConnection class. Access the results in the DataTable returned.

C#

```
DbProviderFactory provider =
DbProviderFactories.GetFactory("System.Data.CompositeClient");
using(DbConnection conn = provider.CreateConnection()) {
  conn.Open();
  DataTable databaseSchema =
conn.GetSchema("ConnectionProperties");
```

```
  foreach (DataRow row in databaseSchema.Rows) {
    Console.WriteLine(row["Name"]);
    Console.WriteLine(row["Type"]);
    Console.WriteLine(row["ShortDescription"]);
  }
}
```

VB.NET

```
Dim provider =
DbProviderFactories.GetFactory("System.Data.CompositeClient")
Using conn As DbConnection = provider.CreateConnection()
  conn.Open()
  Dim databaseSchema As DataTable =
conn.GetSchema("ConnectionProperties")
    For Each row As DataRow In databaseSchema.Rows
    Console.WriteLine(row("Name"))
    Console.WriteLine(row("Type"))
    Console.WriteLine(row("ShortDescription"))
  Next
End Using
```

Columns Returned

The ConnectionProperties schema collection contains the following information:

Column Name	Data Type	Description
Name	System.String	The name of the connection property.
ShortDescription	System.String	A description of the connection property.
Type	System.String	The data type.
Values	System.String	The allowed values.
Default	System.String	The default value if one is not set by the user.
Category	System.String	A category grouping associated connection properties.
Required	System.String	Whether the property is required to connect.

Column Name	Data Type	Description
Value	System.String	The current value of the connection property.

Result Sets

You can access the same column information about the results of a query that you can for table schemas. See Columns for the columns returned.

Retrieving Result Set Metadata

You can use the GetSchemaTable method of the CompositeDataReader to retrieve result set metadata. Call GetSchemaTable after calling ExecuteReader.

Each row of the DataTable describes a column in the query's result.

C#

```
string connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";

using (CompositeConnection conn = new
CompositeConnection(connectionString)) {
  CompositeCommand cmd = new CompositeCommand("SELECT * FROM
Products WHERE ProductName = 'Konbu'", conn);
  CompositeDataReader rdr = cmd.ExecuteReader();
  DataTable schemaTable = rdr.GetSchemaTable();
  foreach (DataRow row in schemaTable.Rows) {
    foreach (DataColumn col in schemaTable.Columns) {
      Console.WriteLine("{0}: {1}", col.ColumnName, row[col]);
    }
  }
}
```

VB.NET

```
Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"
```

```
Using conn As New CompositeConnection(connectionString)
  Dim cmd As New CompositeCommand("SELECT * FROM Products WHERE
ProductName = 'Konbu'", conn)
  Dim rdr As CompositeDataReader = cmd.ExecuteReader()
  Dim schemaTable As DataTable = rdr.GetSchemaTable()
  For Each row As DataRow In schemaTable.Rows
    For Each col As DataColumn In schemaTable.Columns
      Console.WriteLine("{0}: {1}", col.ColumnName, row(col))
    Next
  Next
End Using
```

Connection String Options

The connection string properties are the various options that can be used to establish a connection. This section provides a complete list of the options you can configure in the connection string for this provider. Click the links for further details.

For more information on connecting, see Establishing a Connection.

Alternate Security Credentials	Uses the URL to set an alternate security credentials value for client authorization when using TDV with a client restricted license.
Case Sensitive	Specifies case sensitivity in the request values.
Catalog	The name of the catalog to use.
Commit Failure	Specifies the behavior if a commit fails.
Commit Interrupt	Specifies the behavior if a commit is interrupted.
Compensate	The correcting behavior.
Connection Life Time	The maximum lifetime of a connection in seconds. Once the time has elapsed, the connection object is disposed.
Connect Timeout	The time-out for initial connection, in seconds.
Data Source	The name of the TDV data source.
Default Catalog	The default catalog for a specified connection.

Default Schema	The default schema for a specified connection.
Domain	The TDV domain to which the DataSource belongs.
Enable Fast Exec	Specifies whether to enable fast execution of queries.
Enable Flood	Values are true or false. Default value is true. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.
Enable Reconnect On Error	Specifies cluster reconnection behavior.
Encrypt	Specifies whether to encrypt the connection using SSL.
Fetch Bytes	The maximum number of rows to fetch for a batch based on batch size, in bytes.
Fetch Rows	Maximum number of rows to fetch for a batch.
Host	The name of the server running TDV Server.
Ignore Trailing Spaces	Specifies whether to ignore trailing spaces at the end of values.
Kerberos KDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
Kerberos Realm	The Kerberos Realm used to authenticate the user with.
Kerberos SPN	The Service Principal Name for the Kerberos Domain Controller.
Locale	Value that defines the user's language and country.
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.
Logfile	A path to the log file.
Max Log File Count	A string specifying the maximum file count of log files. When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted.
Max Log File Size	A string specifying the maximum size in bytes for a log file (for example, 10 MB). When the limit is hit, a new log is created in the same folder with the date and time appended to the end.

Max Rows	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
No Metadata	Blocks return of result-set metadata during query execution.
Optimization Prepare	Specifies whether to optimize prepare requests sent to TDV.
Other	These hidden properties are used only in specific use cases.
Param Mode	Controls the behavior of OUT parameters for stored procedures.
Password	The user's password.
Pool Idle Timeout	The allowed idle time for a connection before it is closed.
Pool Max Size	The maximum connections in the pool.
Pool Min Size	The minimum number of connections in the pool.
Pool Wait Time	The max seconds to wait for an available connection.
Port	The port of the TDV server.
Query Passthrough	This option passes the query to the TDV server as is.
Readonly	You can use this property to enforce read-only access to TDV from the provider.
Register Output Cursors	Specifies how to handle output cursors.
Request Timeout	The time-out for query commands and other requests, in seconds.
Session Timeout	Session inactivity time-out, in seconds.
Session Token	Uses the URL to set a session token value for client authorization when using TDV with a client restricted license.
SSL Client Cert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).
SSL Client Cert Password	The password for the TLS/SSL client certificate.

SSL Client Cert Subject	The subject of the TLS/SSL client certificate.
SSL Client Cert Type	The type of key store containing the TLS/SSL client certificate.
SSL Server Cert	The certificate to be accepted from the server when connecting using TLS/SSL.
SSO	The single-sign-on (SSO) type to use to authenticate.
Strip Duplicates	Values are true or false. Default value is true. If true, the server will detect duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire.
Strip Trailing Zeros	Determines whether decimal result values are to be returned with trailing zeroes removed.
Tables	This property restricts the tables reported to a subset of the available tables. For example, Tables=TableA,TableB,TableC.
Trace Folder	The absolute directory to save the trace file.
Trace Level	The level of information to log.
Use Connection Pooling	This property enables connection pooling.
User	The username provided for authentication with TDV Server.
User Tokens	Authentication values that can be packaged for delivery.
Validate Remote Cert	Values are true or false. Default value is false. If true, the client will validate the server's cert.
Validate Remote Hostname	Values are true or false. Default value is false. If true, the client will validate the server's hostname.
Verbosity	The verbosity level that determines the amount of detail included in the log file.
Views	Restricts the views reported to a subset of the available tables. For example, Views=ViewA,ViewB,ViewC.

Alternate Security Credentials

Uses the URL to set an alternate security credentials value for client authorization when using TDV with a client restricted license.

Data Type

string

Default Value

""

Remarks

Uses the URL to set an alternate security credentials value for client authorization when using TDV with a client restricted license.

Case Sensitive

Specifies case sensitivity in the request values.

Data Type

bool

Default Value

false

Remarks

Specifies case sensitivity in the request values. By default (false), requests are not case-sensitive.

Catalog

The name of the catalog to use.

Data Type

string

Default Value

""

Remarks

This field allows you to limit the Catalog to the one explicitly specified. If not set, the provider will retrieve the available catalogs from the TDV server.

Commit Failure

Specifies the behavior if a commit fails.

Data Type

string

Default Value

""

Remarks

Specifies the behavior if a commit fails. Possible values are: rollback or bestEffort.

Commit Interrupt

Specifies the behavior if a commit is interrupted.

Data Type

string

Default Value

""

Remarks

Specifies the behavior if a commit is interrupted. Possible values are: ignore, log, fail.

Compensate

The correcting behavior.

Data Type

string

Default Value

"disabled"

Remarks

The correcting behavior, possible values are: disabled or enabled.

Connection Life Time

The maximum lifetime of a connection in seconds. Once the time has elapsed, the connection object is disposed.

Data Type

string

Default Value

"0"

Remarks

The maximum lifetime of a connection in seconds. Once the time has elapsed, the connection object is disposed. The default is 0 which indicates there is no limit to the connection lifetime.

Connect Timeout

The time-out for initial connection, in seconds.

Data Type

int

Default Value

0

Remarks

This property was added for AWS Data Pipeline compatibility. This tool appends this connection property to the URL and by adding this as a hidden property, we avoid the validation error. Has no functional effect in the driver.

The time-out for initial connection, in seconds. Use 0 (zero) for infinite time-out.

Data Source

The name of the TDV data source.

Data Type

string

Default Value

""

Remarks

Data source refers to the TDV database name published in the Data Services node.

Default Catalog

The default catalog for a specified connection.

Data Type

string

Default Value

""

Remarks

The default catalog for a specified connection.

Default Schema

The default schema for a specified connection.

Data Type

string

Default Value

""

Remarks

The default schema for a specified connection.

Domain

The TDV domain to which the DataSource belongs.

Data Type

string

Default Value

""

Remarks

The TDV domain to which the DataSource belongs.

Typically the domain is 'composite' for installations with locally defined users.

Enable Fast Exec

Specifies whether to enable fast execution of queries.

Data Type

bool

Default Value

false

Remarks

Values are true or false (default).

Results are processed and returned immediately (instead of round trip) when a query is submitted, potentially improving performance of low latency queries.

Enable Flood

Values are true or false. Default value is true. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.

Data Type

bool

Default Value

true

Remarks

Values are true or false. Default value is true. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.

Enable Reconnect On Error

Specifies cluster reconnection behavior.

Data Type

bool

Default Value

false

Remarks

Specifies cluster reconnection behavior.

Encrypt

Specifies whether to encrypt the connection using SSL.

Data Type

bool

Default Value

false

Remarks

When set to true, automatically passes messages to the SSL port for processing with the TDV SSL Certificate.

Fetch Bytes

The maximum number of rows to fetch for a batch based on batch size, in bytes.

Data Type

int

Default Value

131072

Remarks

The maximum number of rows to fetch for a batch based on batch size, in bytes.

Setting FetchBytes to a very large number can cause an Out Of Memory error in the server. The value set for FetchBytes affects the memory used on the client and the TDV server, so the value should be set based on the heap size configured.

Fetch Rows

Maximum number of rows to fetch for a batch.

Data Type

int

Default Value

500

Remarks

Maximum number of rows to fetch for a batch. Set to 0 (zero) to return an unlimited number of rows.

Host

The name of the server running TDV Server.

Data Type

string

Default Value

""

Remarks

This property should be set to the name or network address of the computer running TDV Server.

Ignore Trailing Spaces

Specifies whether to ignore trailing spaces at the end of values.

Data Type

bool

Default Value

false

Remarks

Specifies whether to ignore trailing spaces at the end of values.

Kerberos KDC

The Kerberos Key Distribution Center (KDC) service used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using Windows Authentication. The provider will request session tickets and temporary session keys from the Kerberos Key Distribution Center (KDC) service. The Kerberos Key Distribution Center (KDC) service is conventionally colocated with the domain controller. If Kerberos KDC is not specified the provider will attempt to detect these properties automatically from the following locations:

- **Java System Properties**: Kerberos settings can be configured in Java using the config file krb5.conf, or using the system properties java.security.krb5.realm and java.security.krb5.kdc. The provider will use the system settings if KerberosRealm and KerberosKDC are not explicitly set.
- **Domain Name and Host**: The provider will infer the Kerberos Realm and Kerberos KDC from the configured domain name and host as a last resort.

Note: Windows authentication is supported in JRE 1.6 and above only.

Kerberos Realm

The Kerberos Realm used to authenticate the user with.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using SPNEGO or Windows Authentication. The Kerberos Realm is used to authenticate the user with the Kerberos Key Distribution Service (KDC). The Kerberos Realm can be configured by an administrator to be any string, but conventionally it is based on the domain name. If Kerberos Realm is not specified the provider will attempt to detect these properties automatically from the following locations:

- **Java System Properties**: Kerberos settings can be configured in Java using a config file (krb5.conf) or using the system properties java.security.krb5.realm and java.security.krb5.kdc. The provider will use the system settings if KerberosRealm and KerberosKDC are not explicitly set.
- **Domain Name and Host**: The provider will infer the Kerberos Realm and Kerberos KDC from the user-configured domain name and host as a last resort. This might work in some Windows environments.

Note: Kerberos-based authentication is supported in JRE 1.6 and above only.

Kerberos SPN

The Service Principal Name for the Kerberos Domain Controller.

Data Type

string

Default Value

""

Remarks

If the Service Principal Name on the Kerberos Domain Controller is not the same as the URL that you are authenticating to, set the Service Principal Name here.

Locale

Value that defines the user's language and country.

Data Type

string

Default Value

""

Remarks

Value that defines the user's language and country.

Location

A path to the directory that contains the schema files defining tables, views, and stored procedures.

Data Type

string

Default Value

"%APPDATA%\\TIBCO\\Composite Data Provider\\Schema"

Remarks

The path to a directory which contains the schema files for the provider (.rsd files for tables and views, .rsb files for stored procedures). The folder location can be a relative path from the location of the executable. The Location property is only needed if you want to customize definitions (for example, change a column name, ignore a column, and so on) or extend the data model with new tables, views, or stored procedures.

If left unspecified, the default location is "%APPDATA%\\TIBCO\\Composite Data Provider\\Schema" with %APPDATA% being set to the user's configuration directory:

Platform	%APPDATA%
Windows	The value of the APPDATA environment variable
Mac	~/.config
Linux	~/.config

Logfile

A path to the log file.

Data Type

string

Default Value

""

Remarks

For more control over what is written to the log file, you can adjust its Verbosity.

Max Log File Count

A string specifying the maximum file count of log files. When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted.

Data Type

string

Default Value

""

Remarks

A string specifying the maximum file count of log files. When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted. The minimum supported value is 2. A value of 0 or a negative value indicates no limit on the count.

Max Log File Size

A string specifying the maximum size in bytes for a log file (for example, 10 MB). When the limit is hit, a new log is created in the same folder with the date and time appended to the end.

Data Type

string

Default Value

"100MB"

Remarks

A string specifying the maximum size in bytes for a log file (for example, 10 MB). When the limit is hit, a new log is created in the same folder with the date and time appended to the end. The default limit is 100 MB. Values lower than 100 kB will use 100 kB as the value instead.

Max Rows

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

Data Type

string

Default Value

"-1"

Remarks

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

No Metadata

Blocks return of result-set metadata during query execution.

Data Type

bool

Default Value

false

Remarks

Blocks return of result-set metadata during query execution.

Optimization Prepare

Specifies whether to optimize prepare requests sent to TDV.

Data Type

bool

Default Value

true

Remarks

When set to "True" (default), the provider will submit the query in a single request to TDV.

When set to "False", the provider will submit an initial prepare request to TDV.

Other

These hidden properties are used only in specific use cases.

Data Type

string

Default Value

""

Remarks

The properties listed below are available for specific use cases. Normal driver use cases and functionality should not require these properties.

Specify multiple properties in a semicolon-separated list.

Integration and Formatting

DefaultColumnSize	Sets the default length of string fields when the data source does not provide column length in the metadata. The default value is 2000.
ConvertDateTimeToGMT	Determines whether to convert date-time values to GMT, instead of the local time of the machine.
RecordToFile=filename	Records the underlying socket data transfer to the specified file.

Param Mode

Controls the behavior of OUT parameters for stored procedures.

Data Type

string

Default Value

"normal"

Remarks

Controls the behavior of OUT parameters for stored procedures.

Valid values are:

normal	Report OUT parameters in procedure metadata as OUT parameters.
return	Report OUT parameters as return values.
omit	Omit OUT parameters from metadata.
omitCursors	Omit output cursors from metadata.

Password

The user's password.

Data Type

string

Default Value

""

Remarks

The password provided for authentication with the TDV Server.

Pool Idle Timeout

The allowed idle time for a connection before it is closed.

Data Type

string

Default Value

""

Remarks

The allowed idle time a connection can remain in the pool until the connection is closed. The default is 60 seconds.

Pool Max Size

The maximum connections in the pool.

Data Type

string

Default Value

"100"

Remarks

The maximum connections in the pool. The default is 100. To disable this property, set the property value to 0 or less.

Pool Min Size

The minimum number of connections in the pool.

Data Type

string

Default Value

"1"

Remarks

The minimum number of connections in the pool. The default is 1.

Pool Wait Time

The max seconds to wait for an available connection.

Data Type

string

Default Value

""

Remarks

The max seconds to wait for a connection to become available. If a new connection request is waiting for an available connection and exceeds this time, an error is thrown. By default, new requests wait forever for an available connection.

Port

The port of the TDV server.

Data Type

int

Default Value

9401

Remarks

The port of the server hosting the TDV Server.

Default is 9401 (plaintext) and the SSL protected port is 9403.

Query Passthrough

This option passes the query to the TDV server as is.

Data Type

bool

Default Value

false

Remarks

When this is set, queries are passed through directly to TDV.

Readonly

You can use this property to enforce read-only access to TDV from the provider.

Data Type

bool

Default Value

false

Remarks

If this property is set to true, the provider will allow only SELECT queries. INSERT, UPDATE, DELETE, and stored procedure queries will cause an error to be thrown.

Register Output Cursors

Specifies how to handle output cursors.

Data Type

bool

Default Value

false

Remarks

Specifies how to handle output cursors.

Valid values are:

true	Bind or register output cursors as output parameters.
false	Do not bind or register output cursors as output parameters; instead, use SQLMoreResults or Statement.getMoreResults() to access the cursors.

Request Timeout

The time-out for query commands and other requests, in seconds.

Data Type

int

Default Value

0

Remarks

The time-out for query commands and other requests, in seconds.

Session Timeout

Session inactivity time-out, in seconds.

Data Type

int

Default Value

0

Remarks

Session inactivity time-out, in seconds. Set to 0 (zero) for infinite time-out.

Session Token

Uses the URL to set a session token value for client authorization when using TDV with a client restricted license.

Data Type

string

Default Value

""

Remarks

Uses the URL to set a session token value for client authorization when using TDV with a client restricted license.

SSL Client Cert

The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).

Data Type

string

Default Value

""

Remarks

The name of the certificate store for the client certificate.

The SSLClientCertType field specifies the type of the certificate store specified by SSLClientCert. If the store is password protected, specify the password in SSLClientCertPassword.

SSLClientCert is used in conjunction with the SSLClientCertSubject field in order to specify client certificates. If SSLClientCert has a value, and SSLClientCertSubject is set, a search for a certificate is initiated. See SSL Client Cert Subject for more information.

Designations of certificate stores are platform-dependent.

The following are designations of the most common User and Machine certificate stores in Windows:

MY	A certificate store holding personal certificates with their associated private keys.
CA	Certifying authority certificates.
ROOT	Root certificates.
SPC	Software publisher certificates.

In Java, the certificate store normally is a file containing certificates and optional private keys.

When the certificate store type is PFXFile, this property must be set to the name of the file. When the type is PFXBlob, the property must be set to the binary contents of a PFX file (for example, PKCS12 certificate store).

SSL Client Cert Password

The password for the TLS/SSL client certificate.

Data Type

string

Default Value

""

Remarks

If the certificate store is of a type that requires a password, this property is used to specify that password to open the certificate store.

SSL Client Cert Subject

The subject of the TLS/SSL client certificate.

Data Type

string

Default Value

"*"

Remarks

When loading a certificate the subject is used to locate the certificate in the store.

If an exact match is not found, the store is searched for subjects containing the value of the property. If a match is still not found, the property is set to an empty string, and no certificate is selected.

The special value "*" picks the first certificate in the certificate store.

The certificate subject is a comma separated list of distinguished name fields and values. For example, "CN=www.server.com, OU=test, C=US, E=support@company.com". The common fields and their meanings are shown below.

Field	Meaning
CN	Common Name. This is commonly a host name like www.server.com.
O	Organization
OU	Organizational Unit
L	Locality
S	State

Field	Meaning
C	Country
E	Email Address
If a field value contains a comma, it must be quoted.	

SSL Client Cert Type

The type of key store containing the TLS/SSL client certificate.

Data Type

string

Default Value

""

Remarks

This property can take one of the following values:

USER - default	For Windows, this specifies that the certificate store is a certificate store owned by the current user. Note that this store type is not available in Java.
MACHINE	For Windows, this specifies that the certificate store is a machine store. Note that this store type is not available in Java.
PFXFILE	The certificate store is the name of a PFX (PKCS12) file containing certificates.
PFXBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in PFX (PKCS12) format.
JKSFILE	The certificate store is the name of a Java key store (JKS) file containing certificates. Note that this store type is only available in Java.

JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in JKS format. Note that this store type is only available in Java.
PEMKEY_FILE	The certificate store is the name of a PEM-encoded file that contains a private key and an optional certificate.
PEMKEY_BLOB	The certificate store is a string (base64-encoded) that contains a private key and an optional certificate.
PUBLIC_KEY_FILE	The certificate store is the name of a file that contains a PEM- or DER-encoded public key certificate.
PUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains a PEM- or DER-encoded public key certificate.
SSHPUBLIC_KEY_FILE	The certificate store is the name of a file that contains an SSH-style public key.
SSHPUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains an SSH-style public key.
P7BFILE	The certificate store is the name of a PKCS7 file containing certificates.
PPKFILE	The certificate store is the name of a file that contains a PuTTY Private Key (PPK).
XMLFILE	The certificate store is the name of a file that contains a certificate in XML format.
XMLBLOB	The certificate store is a string that contains a certificate in XML format.

SSL Server Cert

The certificate to be accepted from the server when connecting using TLS/SSL.

Data Type

string

Default Value

""

Remarks

If using a TLS/SSL connection, this property can be used to specify the TLS/SSL certificate to be accepted from the server. Any other certificate that is not trusted by the machine is rejected.

This property can take the following forms:

Description	Example
A full PEM Certificate (example shortened for brevity)	-----BEGIN CERTIFICATE----- MIIChTCCAe4CAQAwDQYJKoZIhv......Qw== -----END CERTIFICATE-----
A path to a local file containing the certificate	C:\cert.cer
The public key (example shortened for brevity)	-----BEGIN RSA PUBLIC KEY----- MIGfMA0GCSq......AQAB -----END RSA PUBLIC KEY-----
The MD5 Thumbprint (hex values can also be either space or colon separated)	ecadbdda5a1529c58a1e9e09828d70e4
The SHA1 Thumbprint (hex values can also be either space or colon separated)	34a929226ae0819f2ec14b4a3d904f801cbb150d

If not specified, any certificate trusted by the machine is accepted.

Use '*' to signify to accept all certificates. Note that this is not recommended due to security concerns.

SSO

The single-sign-on (SSO) type to use to authenticate.

Data Type

string

Default Value

"Disable"

Remarks

The single-sign-on (SSO) type to use to authenticate. Valid values are: Disable, Kerberos, and NTLM.

Valid on Windows platform only.

Default is "Disable" which forces the client to provide a user and password to authenticate.

Strip Duplicates

Values are true or false. Default value is true. If true, the server will detect duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire.

Data Type

bool

Default Value

true

Remarks

Values are true or false. Default value is true. If true, the server will detect duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire.

Strip Trailing Zeros

Determines whether decimal result values are to be returned with trailing zeroes removed.

Data Type

bool

Default Value

false

Remarks

Determines whether decimal result values are to be returned with trailing zeroes removed.

Tables

This property restricts the tables reported to a subset of the available tables. For example, Tables=TableA,TableB,TableC.

Data Type

string

Default Value

""

Remarks

Listing the tables from some databases can be expensive. Providing a list of tables in the connection string improves the performance of the provider.

This property can also be used as an alternative to automatically listing views if you already know which ones you want to work with and there would otherwise be too many to work with.

Specify the tables you want in a comma-separated list. For example, Tables=TableA,TableB,TableC.

Trace Folder

The absolute directory to save the trace file.

Data Type

string

Default Value

""

Remarks

The absolute directory to save the trace file.

Trace Level

The level of information to log.

Data Type

string

Default Value

"error"

Remarks

The level of information to log. Valid values are: off, fatal, error (default), warn, info, debug, and all.

Use Connection Pooling

This property enables connection pooling.

Data Type

bool

Default Value

false

Remarks

This property enables connection pooling. The default is false. See Connection Pooling for information on using connection pools.

User

The username provided for authentication with TDV Server.

Data Type

string

Default Value

""

Remarks

The username provided for authentication with TDV Server.

User Tokens

Authentication values that can be packaged for delivery.

Data Type

string

Default Value

""

Remarks

Authentication values that can be packaged for delivery.

The URL can pass the user_tokens property to the server at the init command, in the form: " user_tokens=(" NAME "=" VALUE ("," NAME "=" VALUE)* ")"

Validate Remote Cert

Values are true or false. Default value is false. If true, the client will validate the server's cert.

Data Type

bool

Default Value

false

Remarks

Values are true or false. Default value is false. If true, the client will validate the server's cert.

Validate Remote Hostname

Values are true or false. Default value is false. If true, the client will validate the server's hostname.

Data Type

bool

Default Value

false

Remarks

Values are true or false. Default value is false. If true, the client will validate the server's hostname.

Verbosity

The verbosity level that determines the amount of detail included in the log file.

Data Type

string

Default Value

"1"

Remarks

The verbosity level determines the amount of detail that the provider reports to the Logfile. Verbosity levels from 1 to 5 are supported. These are described in the following list:

1	Setting Verbosity to 1 will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.
2	Setting Verbosity to 2 will log everything included in Verbosity 1 and additional information about the request, if applicable.
3	Setting Verbosity to 3 will additionally log the body of the request and the response.
4	Setting Verbosity to 4 will additionally log transport-level communication with the data source. This includes SSL negotiation.
5	Setting Verbosity to 5 will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands.

The Verbosity should not be set to greater than 1 for normal operation. Substantial amounts of data can be logged at higher verbosities, which can delay execution times.

Views

Restricts the views reported to a subset of the available tables. For example, Views=ViewA,ViewB,ViewC.

Data Type

string

Default Value

""

Remarks

Listing the views from some databases can be expensive. Providing a list of views in the connection string improves the performance of the provider.

This property can also be used as an alternative to automatically listing views if you already know which ones you want to work with and there would otherwise be too many to work with.

Specify the views you want in a comma-separated list. For example, Views=ViewA,ViewB,ViewC.

TIBCO SSIS Components for TDV

Overview

The TIBCO SSIS Components for TIBCO(R) Data Virtualization enable you to connect SQL Server with TDV data through SSIS workflows. The components wrap the complexity of accessing TDV data in standard SSIS data flow components. You can then connect and synchronize TDV tables with SQL Server tables.

The components hide the complexity of accessing data and provide additional security features, smart caching, batching, socket management, and more.

Key Features

- Create, read, update, and delete (CRUD) support.
- Access TDV data in real time.
- Integrate TDV data without the need for custom development.

Getting Started

Getting Started covers Establishing a Connection with the Connection Manager and selecting rows Using the Source Component, and making changes Using the Destination Component. See the TDV integration guides for information on connecting from other applications.

Advanced Features

Advanced Features details additional features supported by the component, such as , ssl configuration, firewall/proxy settings, and advanced logging.

See Data Model for the available schema information.

Connection Properties

The Connection Properties describe the various options that can be used to establish a connection.

Getting Started

Connecting to TDV

Establishing a Connection shows how to authenticate to TDV and configure any necessary connection properties from the Connection Manager window.

You can also configure component capabilities through the available Connection Properties, from data modeling to firewall traversal. The Advanced Settings section shows how to set up more advanced configurations and troubleshoot connection errors.

SSIS Version Support

The TIBCO SSIS Components for TIBCO(R) Data Virtualization 2019 provide data flow components that allow for straightforward integration with TDV data from SSIS data flow tasks. SQL Server 2008 R2, 2012, 2014, 2016, 2017, and 2019 are supported including their corresponding version of SQL Server Data Tools.

TDV Version Support

The component enables read/write SQL-92 access to TIBCO(R) Data Virtualization version 7.0.1 and above.

See Also

Using the Source Component: Pull TDV data into your data flow task.

Using the Destination Component: Push data to TDV.

Deploying to Azure: Deploy the TIBCO components to Azure from Visual Studio.

Adding Items to the Toolbox: The component installation automatically adds the components to the toolbox; if you do not see the components, follow this procedure to add them.

Adding Items to the Toolbox

Visual Studio 2012 and Later

In Visual Studio 2012 and later, the components are automatically added to the toolbox.

If you do not see the TIBCO components, ensure you are on the Data Flow tab in the project and check the target SQL Server version of the package. You can find the version in the solution's properties under **Configuration Properties > General > TargetServerVersion**. The components only appear for versions of SQL Server installed on the same machine.

Visual Studio 2008

In Visual Studio 2008, you need to add the TIBCO TDV components to the Toolbox manually by completing the following steps:

1. To add the Source and Destination, right-click anywhere in the SSIS Toolbox and select **Choose Items**.
2. In the Choose Toolbox Items window, select the **SSIS Data Flow Items** tab.
3. From this tab, select the TIBCO TDV components.

The components are then available in the SSIS toolbox on the Data Flow tab.

Establishing a Connection

Connecting with the TDV Connection Manager

Adding a new TDV connection to the SSIS package is straightforward. Right-click within the Connection Manager window and select New Connection from the menu. Then, choose the TDV Connection Manager from the Add SSIS Connection Manager window.

Alternatively, you can create a new connection directly using the TDV Source or the TDV Destination. You can set the connection string options in the TDV Connection Manager window.

Set the Host, Domain, User, Password, and DataSource connection properties to connect to the TDV Server.

Deploying to Azure

With the release of Azure Data Factory V2 integration runtimes (ADFv2 IR), deployment to the Azure cloud is now possible for SSIS projects that use TIBCO components. Follow the steps below to deploy the components to Azure. You can then execute SSIS projects in the configured Azure Data Factory.

You will complete the following tasks:

- Run the AzureDeploy.ps1 script to provision an integration runtime.
- Deploy the SSIS project From Visual Studio to Azure.
- Use SSMS to manage and execute a deployed project.

Prerequisites

In addition to an Azure subscription, you will need the following to deploy the components:

- **SSMS 2012 or higher**
- **Azure resource group**
- **Azure SQL database**

Confirm that the SQL Server's firewall settings will allow access from the client machine: From the SQL Server's overview page, click Set Server Firewall and add the IP address of the client machine, or a range of IP addresses that includes the IP address of the client machine. Additionally, ensure that the Allow access to Azure Services option is enabled in the firewall settings.

- **AzureRM PowerShell 6.2.0 or higher**

You can obtain Azure PowerShell by using PowerShellGet or downloading the .msi installer. After installing, you may need to run the following:

```
Set-ExecutionPolicy Unrestricted
```

Deploying TIBCO SSIS Components to Azure

Follow the procedure below to deploy SSIS projects using one or more TIBCO data sources:

Provision an Integration Runtime

Use the included AzureDeploy.ps1 script to provision and start an Azure SSIS integration runtime.

1. To deploy multiple TIBCO SSIS components, copy all other TIBCO SSIS2017.dll and .Design.dll files to this component's /lib folder. By default, this is C:\Program Files\TIBCO\TIBCO SSIS Components for TIBCO(R) Data Virtualization\lib.
2. Run the AzureDeploy.ps1 script from this /lib directory. You may specify all parameters required parameters at once. For example:

```
.\AzureDeploy.ps1 -ResourceGroupName "my-resource-group"
-SubscriptionId "2d91834e-1hga-4c31-86yf-dba7b40b90u2"
-SqlServerName "my-sql-db.database.windows.net" -SqlDatabaseUser
"MySQLUser" -SqlDatabasePwd "MySQLPwd" -DataFactoryName
"MyDataFactory" -StorageAccountName "MyStorageAccount"
```

If your SQL Server database already has an integration runtime (and SSISDB), you can overwrite it by first stopping it, and then specifying its name with the -InterationRuntimeName parameter. Each SQL Server may only have one integration runtime.

See the "Configuring the AzureDeploy.ps1 Script" section below for more information on the available parameters.

3. Log into Azure in the dialog that is displayed.

After you log into Azure, the script creates the resources necessary for deployment and starts the integration runtime.

Deploy the SSIS Package

You are now ready to deploy your SSIS package:

- In Visual Studio, right-click the project and select Deploy Project. The Integration Services Deployment Wizard is displayed.
- On the Select Source page, select your SSIS project. You can select a project deployment file or a project that resides in an SSIS catalog.
- On the Select Destination page, enter the fully qualified domain name of the logical SQL database and enter your authentication information. Select Browse to select the target folder in SSISDB.

After this step, your package is deployed and accessible in the Azure Data Factory web UI at https://adf.azure.com.

Managing and Running the Project in SSMS

By default, connection settings with sensitive information (passwords, security tokens, etc.) are redacted when deploying to Azure. To be able to execute projects in the configured Azure Data Factory, provide this information through SSMS.

1. Connect to the Azure SQL database. In Options -> Connection Properties, set the Connect to Database field to "SSISDB".
2. Right-click the project and click Configure to configure the SSIS project in the Integration Services Catalog.

Note: If you do not see Integration Services Catalogs, you may need to upgrade your SSMS version or set "SSISDB" in step 1.

3. Add connection information as necessary in the Connection Manager tab. Provide the RTK connection property.

You can then execute the project. View the results of execution by right-clicking the project and clicking Reports -> All Executions.

Configuring the AzureDeploy.ps1 Script

The following sections provide a reference to the available options for the deployment script:

Required Parameters

- **ResourceGroupName**: The name of the resource group to use or create resources in. The resource group must exist.
- **SqlServerName**: The name or endpoint of the logical SQL database. Example: ssishost.database.windows.net.
- **SqlDatabaseUser**: The username of the SQL Server user.
- **SqlDatabasePwd**: The password for the SQL Server user.
- **DataFactoryName**: The name of the Data Factory to use (or create). A data factory may only have one integration runtime. To overwrite an existing IR, specify its name with the
`-IntegrationRuntimeName parameter.`
- **StorageAccountName**: The name of the storage account to use (or create).

Other Parameters

- **SubscriptionId:** The Id of the subscription to use for creating additional resources. This should match a subscription from ResourceGroupName. Example: 2r29814e-1dba-4b11-81cf-dba7b90b74c3
- **Location:** The location to create additional resources. Defaults to "EastUS".
- **SetupContainerName:** The name of the blob container to create (or reuse). TIBCO assembly files, main.cmd, and SSISDeployUtil.bat will be overwritten if they exist. Defaults to "ssissetup".
- **StorageAccountResourceGroup**: If the storage account to be used is on a different resource group, this parameter specifies the name of the resource group for the -StorageAccountName to use. Defaults to -ResourceGroup otherwise.

- **RuntimeNodeSize:** The integration runtime node size. Defaults to "Standard_D1_v2".
- **RuntimeNodeCount:** The number of target nodes of the integration runtime. Defaults to 1.
- **AzureSSISMaxParallelExecutionsPerNode:** Max Parallel Executions Per Node. Defaults to 1.
- **CatalogPricingTier:** The catalog database pricing tier of the integration runtime. Defaults to "Basic".
- **AzureSSISEdition:** The edition of the SSIS integration runtime. Standard or Enterprise. Defaults to Standard.
- **AzureSSISDescription:** A description you may provide for the Azure SSIS Runtime. Defaults to "Azure SSIS Runtime".
- **IntegrationRuntimeName**: The name of the integration runtime to create (or overwrite). Defaults to "AzureSSISIR".

Each SQL Server can have only one integration runtime. If the existing SQL Server already has an integration runtime and SSISDB (the SSIS catalog database), then you may overwrite it by specifying the
`-InterationRuntimeName parameter.`

Using the AzureLogfile.ps1 Script

The AzureLogfile script assists with mounting an Azure FileShare to both your local system and to your provisioned SSIS IR. As the name suggests, this is useful for troubleshooting with a CData Logfile, but it can also be useful for specifying schema files with the Location property, or for reading/writing to flat files from a deployed package.

Required Parameters

- **ResourceGroupName:** The name of the resource group to use or create resources in. The resource group must exist.
- **StorageAccountName:** The name of the storage account to use.
- **FileShareName:** The name of the file share to use (or create) for mounting.
- **DriveLetter:** The single-character drive letter to mount the fileshare to on the local system
- **SqlServerName:** The name or endpoint of the logical SQL database. Example: ssishost.database.windows.net.
- **SqlDatabaseUser:** The username of the SQL Server user.
- **SqlDatabasePwd:** The password for the SQL Server user.

Changelog

General Changes

[7785] - 2021-04-23

Added

Added support for handling client side formulas during insert / update. For example: UPDATE Table SET Col1 = Concat(Col1, " - ", Col2) WHERE Col2 LIKE 'A%'

[7783] - 2021-04-23

Changed

Updated how display sizes are determined for varchar primary key and foreign key columns so they will match the reported length of the column.

[7776] - 2021-04-16

Added

- Non-conditional updates between two columns is now available to all drivers. For example: UPDATE Table SET Col1=Col2

Changed

- Reduced the length to 255 for varchar primary key and foreign key columns.
- Updated implicit and metadata caching to improve performance and support for multiple connections. Old metadata caches are not compatible - you would need to generate new metadata caches if you are currently using CacheMetadata.
- Updated index naming convention to avoid duplicates
- Updated and standardized Getting Started connection help.
- Added the Advanced Features section to the help of all drivers.
- Categorized connection property listings in the help for all editions.

SSIS Changes

[7776] - 2021-04-16

Added

- Added a create table button for the Destination Component to make it easier to add a new destination in SSIS.
- Improved user experience for Source Component by allowing users to select which columns to select via a new checkbox dialog.

Changed

- Improved query dialog for building parameterized statements.

Advanced Features

This section details a selection of advanced features of the component

SSL Configuration

Use SSL Configuration to adjust how certificate negotiations are handled by the component. You can specify a specific certificate for use in SSL.

Firewall and Proxy

Configure the component for compliance with Firewall and Proxy, including Windows proxies. You can also set up tunnel connections.

Logging

See Logging for an overview of configuration settings that can be used to refine TIBCO logging.

SSL Configuration

Customizing the SSL Configuration

By default, the component attempts to negotiate SSL/TLS by checking the server's certificate against the system's trusted certificate store.

To specify another certificate, see the SSLServerCert property for the available formats to do so.

Client SSL Certificates

The TDV component also supports setting client certificates. Set the following to connect using a client certificate.

- SSLClientCert: The name of the certificate store for the client certificate.
- SSLClientCertType: The type of key store containing the TLS/SSL client certificate.
- SSLClientCertPassword: The password for the TLS/SSL client certificate.
- SSLClientCertSubject: The subject of the TLS/SSL client certificate.

Firewall and Proxy

Connecting Through a Firewall or Proxy

Set the following properties:

- To use a proxy-based firewall, set FirewallType, FirewallServer, and FirewallPort.
- To tunnel the connection, set FirewallType to TUNNEL.
- To authenticate, specify FirewallUser and FirewallPassword.
- To authenticate to a SOCKS proxy, additionally set FirewallType to SOCKS5.

Logging

Capturing component logging can be very helpful when diagnosing error messages or other unexpected behavior.

Basic Logging

You will simply need to set two connection properties to begin capturing component logging.

- Logfile: A filepath which designates the name and location of the log file.
- Verbosity: This is a numerical value (1-5) that determines the amount of detail in the log. See the page in the Connection Properties section for a breakdown of the five levels.

- MaxLogFileSize: When the limit is hit, a new log is created in the same folder with the date and time appended to the end. The default limit is 100 MB. Values lower than 100 kB will use 100 kB as the value instead.
- MaxLogFileCount: A string specifying the maximum file count of log files. When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted. Minimum supported value is 2. A value of 0 or a negative value indicates no limit on the count.

Once this property is set, the component will populate the log file as it carries out various tasks, such as when authentication is performed or queries are executed. If the specified file doesn't already exist, it will be created.

Log Verbosity

The verbosity level determines the amount of detail that the component reports to the Logfile. Verbosity levels from 1 to 5 are supported. These are described in the following list:

1. Setting Verbosity to 1 will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.
2. Setting Verbosity to 2 will log everything included in Verbosity 1 and additional information about the request, if applicable.
3. Setting Verbosity to 3 will additionally log the body of the request and the response.
4. Setting Verbosity to 4 will additionally log transport-level communication with the data source. This includes SSL negotiation.
5. Setting Verbosity to 5 will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands.

The Verbosity should not be set to greater than 1 for normal operation. Substantial amounts of data can be logged at higher verbosities, which can delay execution times.

To refine the logged content further by showing/hiding specific categories of information, see LogModules.

Advanced Logging

You may want to refine the exact information that is recorded to the log file. This can be accomplished using the LogModules property.

This property allows you to filter the logging using a semicolon-separated list of logging modules.

All modules are four characters long. **Please note that modules containing three letters have a required trailing blank space.** The available modules are:

- EXEC: Query Execution. Includes execution messages for original SQL queries, parsed SQL queries, and normalized SQL queries. Query and page success/failure messages appear here as well.
- INFO: General Information. Includes the connection string, driver version (build number), and initial connection messages.
- HTTP: HTTP Protocol messages. Includes HTTP requests/responses (including POST messages), as well as Kerberos related messages.
- SSL : SSL certificate messages.
- OAUTH: OAuth related failure/success messages.
- SQL : Includes SQL transactions, SQL bulk transfer messages, and SQL result set messages.
- META: Metadata cache and schema messages.
- TCP : Incoming and Ongoing raw bytes on TCP transport layer messages.

An example value for this property would be.
```
LogModules=INFO;EXEC;SSL ;SQL ;META;
```

Note that these modules refine the information as it is pulled after taking the Verbosity into account.

Using the SSIS Components

Reading and Writing TDV Data with the SSIS Components

Using the Source Component

Using the Source Component details how to add a source component to the package, including defining a connection manager and selecting a data access mode.

Using the Destination Component

Using the Destination Componentdetails how to add a destination component to the package, how to connect the source and destination components, and selecting data operations to be done with the source data.

Using the Lookup Component

Using the Lookup Component demonstrates how to perform data lookups on a specified set of columns from the source component in a lookup component.

Using the Execute SQL Task

Using the Execute SQL Task details the process for adding an Execute SQL task to your project and executing SQL queries directly.

Calling Stored Procedures

Calling Stored Procedures describes the process for executing stored procedures with a Script component.

Using the Source Component

After Establishing a Connection to the data source, you can use the TIBCO TDV source component to pull data into your Data Flow task.

Querying TDV Data with the Source Component

Follow the procedure below to connect to TDV, retrieve data, and provide data to other components in the workflow.

1. In the SSIS Toolbox, drag the TIBCO TDV source component into the Data Flow task.
2. Double-click the TIBCO TDV source component. The TIBCO TDV Source Editor will display.
3. In the Connection Managers menu, select an available TIBCO TDV connection manager, or create a new instance if one is not already available.
4. Choose your Access Mode: "Table or View" or "SQL Statement". Select "Table or View" to use the GUI to select a table or view. Select "SQL Statement" to configure a statement of your choice.
5. Select the Columns tab and rename any output columns as desired.

When you execute the data flow, rows from your selected table or statement will be made available to the components in the data flow.

Using Parameterized Queries

Parameterized statements provide an efficient way to execute queries dynamically and mitigate SQL injection attacks. The Source Component provides a Parameters button that can be used to map parameters defined in the query to variables in the data flow when using a custom SQL Command. The component will execute these queries as parameterized statements at runtime.

In order to use the Parameters option, your query must contain parameters, which can be either defined positionally as a named prarameter:

- **Positional parameters**: When setting up the parameter mapping, the names in the Parameter list must be the index (starting from 1) of that parameter in the query. Example query:

```
SELECT * FROM Table WHERE FirstName = ? AND Date > ?
```

Here, the Parameter names must be set to '1' and '2' for 'FirstName' and 'Date' respectively.

- **Named parameters**: When setting up the parameter mapping, the names in the Parameter list must exactly match the names of the parameters in the query without the preceding '@' symbol. Example query:

```
SELECT * FROM Table WHERE FirstName = @FirstName AND Date > @Date
```

Here, the Parameter names must be set to 'FirstName' and 'Date'.

Building Parameterized Queries in the Expression Builder

After configuring a source component, you can then use the SSIS Expression Builder to access the SQL statement that the source component executes at run time.

The component will execute these queries as parameterized statements. Parameterized statements provide an efficient way to execute similar queries and mitigate SQL injection attacks.

1. In SSIS Designer, click the Control Flow tab.
2. In the Properties pane, click the button in the box for the Expressions property.
3. In the resulting Property Expressions Editor, click an empty row in the Property box and select the SQLStatement property of the TIBCO TDV source component from the drop-down menu. Then click the button in the row you just added. This displays the Expression Builder.
4. In the Expression box, you can create new SQL commands that use the variables available at run time as input parameters. Ensure that you enclose the expression in quotes. For example:

```
"SELECT * FROM Table WHERE FirstName = '" + @[User::Name] + "' AND
Date > '" + (DT_WSTR, 50) DATEADD("day", -30, GETDATE()) + "'"
```

Using the Destination Component

After Establishing a Connection to the data source, add the TIBCO TDV destination component to the workflow to load data into TDV.

Writing to TDV in a Data Flow

Follow the steps below to connect to TDV and update data.

1. In the SSIS Toolbox, drag the TIBCO TDV destination component into the Data Flow Task.
2. Connect the output of a source component to the TIBCO TDV destination component.
3. Double-click the TIBCO TDV destination component. The TIBCO TDV Destination Editor dialog will display.
4. In the Connection Managers menu, select an available TIBCO TDV connection manager, or create a new instance if one is not already available.
5. In the "Use a Table" option, select the table to update.
6. Select the data manipulation action. See below for more information on each action.
7. On the Mappings tab, configure the mappings from source to destination. By default, outputs from the source component will automatically be mapped with the same name as the columns in the table you selected. You can further update these selections.

Note: Read-only columns will not be visible among the destination columns since they cannot be written to.

Command Execution

When you execute the data flow, the component will execute one of the following operations to update the destination table.

Insert

The component will take the mapped values and attempt to insert the data as new rows into the table. By setting the OutputKey property to True in the destination component's properties, you can retrieve the results of the insert in the error output of the component with the 'Redirect row' error behavior.

Update

The component will attempt to update an existing row based on the primary key provided. The primary key column must be mapped, and it must not be null. By setting the OutputKey property to True in the destination component's properties, you can retrieve the results of the update in the error output of the component with the 'Redirect row' error behavior.

Upsert

The component uses the primary key to decide if a row is to be inserted or updated. If the primary key column is mapped and it is not null, the component will attempt to update an existing row based on the primary key provided. If the primary key is not mapped or if it is null, the TIBCO TDV Destination Component will attempt to insert the data as a new row. By setting the OutputKey property to True in the destination component's properties, you can retrieve the results of the upsert in the error output of the component with the 'Redirect row' error behavior.

Delete

The component will attempt to delete an existing row based on the primary key provided. The primary key column must be mapped, and it must not be null.

Bulk Operations

The destination component will by default use bulk operations to update the data source. This behavior is controlled by the BatchMode and BatchSize properties of the component. The BatchSize controls the maximum size of the batches to submit to the component at once. Depending on the volume of data being submitted, increasing the BatchSize can improve throughput but will require a larger memory footprint.

Using the Lookup Component

After Establishing a Connection to the data source, you can use the TIBCO TDV lookup component to do a lookup against TDV and map the matched and unmatched rows to different outputs in your Data Flow task.

Performing a lookup to TDV Data with the Lookup Component

You can use the Lookup Component to perform a lookup on a specified set of columns in TDV. This component takes one input and has two outputs: one for matched rows in TDV and one for unmatched rows. The component provides two cache options which can affect the performance of the lookup but do not change the results of the lookup operation. These cache options are described later in this section.

1. In the SSIS Toolbox, drag the TIBCO TDV Lookup Component into the Data Flow.
2. Double-click the TIBCO TDV Lookup Component. The TIBCO TDV Lookup Component opens.
3. Navigate to the Connection tab. In the Connection drop-down list, select an available TIBCO TDV connection manager, or create a new instance if one is not already available.
4. Choose your Access Mode: "Table or View" or "SQL Statement". Select "Table or View" to use the GUI to select a table or view. Select "SQL Statement" to configure a statement of your choice.
5. Next, you need to configure fields you wish to use to perform the lookup on. Click on a column in the Available Input Columns list and drag it to the column you want to look up against in the Available Lookup Columns box.
6. Optionally, you can select fields you would like to include in the outputs by checking the boxes for each column in the Available Lookup Columns section. You have the option to add the field to your output as a new field or to overwrite one of the fields in the input.

Full Cache

The default cache mode for the Lookup Component is Full Cache. With this mode, the Lookup Component creates a temporary SQLite cache file as soon as the first input row is detected. The cache is populated with all the rows in the TDV table but only the columns that are selected in the UI. Once the cache is built, each row in the input is looked up against this local cache. This mode is optimal when working with large sets of data that require many lookups.

Partial Cache

Partial Cache mode is similar to Full Cache, but instead of pulling the entire table initially, the Lookup Component batches 100 rows from the input and issue a query to the data source in the following form and cache only those results:

```
SELECT ... WHERE LookupField IN ('value1', ..., 'value100')
```

This mode is only possible if looking up a single column. This mode can improve performance if you're working with a small set of input data where caching the entire TDV table is very expensive.

Using the Execute SQL Task

After Establishing a Connection to the data source, you can use the TIBCO TDV Task run stored procedures and SQL queries at the Control Flow level.

Querying TDV Data with the TDV Task

Complete the following steps to connect to TDV and execute custom SQL queries at the Control Flow level. A common use case for this would be to truncate table data or executing stored procedures before executing your workflow.

1. In the SSIS Toolbox, drag the TIBCO TDV Task into the Control Flow.
2. Double-click the TIBCO TDV Task. The TIBCO TDV Task opens.
3. In the Connection drop-down list, select an available TIBCO TDV connection manager, or create a new instance if one is not already available.
4. Configure the query you want to execute. There are three properties that control this:

- **SQLSourceType:** Where to source the query from, either a direct input in the TIBCO TDV Task UI or from a variable in the package.
- **CommandType:** The type of command to run. The options are Table, Stored Procedure, or Command Text. For the first two, the Task expects only the name of the object to execute. The Command Text option allows you to execute any SQL command you'd like.
- The last option is context-specific based on the values you set for the above two settings.

5. Optionally, you may assign parameters and/or a result set, as described in the following section.

Defining the Parameter Mapping

Parameters can be used with the Stored Procedure and Command Text CommandTypes. On the Parameter Mapping tab, add as many Parameters as you need. When using the Stored Procedure option, the Parameter Name must exactly match the name of the parameter defined in the stored procedure. For the Command Text option, the name must match the name used in the SQL command. You can assign a variable to either read from or write to depending on the Direction chosen. The available Direction options are as follows:

- **Input:** Indicates the value is read from an SSIS variable when executing the call.
- **Output:** Indicates the value is returned from the command call and stored in a local variable for use later in the package.
- **InOut:** A combination of the above. The value is read when the query is executed and the returned value overwrites the existing value after execution is complete.

Configuring a Result Set

A Result Set can be configured if the query results are needed later in the package. There are two options: Single Row and Full ResultSet. Each one affects how the values are output by the Task and how you need to define the output on the Result Set tab in the UI. Note that when using a Stored Procedure, defining OUT or IN/OUT parameters can be used instead of configuring a Result Set.

- **Single Row:** When a single row is returned, you need to set the ResultSet Name to the index (i.e. 0, 1, 2, etc.) of the column you are querying. The Variable Name must be mapped to a variable in SSIS that is the correct data type as the output column.
- **Full ResultSet:** With the Full ResultSet option, only a single value is returned which is a DataTable containing the full data returned from the query. The ResultSet Name should be set to 0. The SSIS Variable Name must be set to a variable of type Object. Note that if multiple ResultSets are returned from the query, the Task currently only supports outputting the first.

Calling Stored Procedures

You can execute stored procedures in a source component as well as in a script component.

Call Stored Procedures from a Source Component

Follow the steps below to execute a stored procedure with the SQL EXEC keyword:

1. Double-click the component in the Data Flow task to open the editor.
2. In the Data Access Mode menu, select SQL Command. The query syntax for stored procedure statements follows the standard form, shown below:

```
EXECUTE my_proc @first = '1', @second = '2', @third = '3';

EXEC my_proc @first = '1', @second = '2', @third = '3';
```

EXECUTE and EXEC can be used interchangeably. See Using the Source Component for how to parameterize the query.

Call Stored Procedures from a Script Component

The following sections show how to call stored procedures in a script component.

Add the Script Component

Add a script component to the data flow from the toolbox and select the type of script component:

- A source script component will have only outputs.
- A destination script component will accept only inputs.
- A transformation script component will accept input columns and produce output columns.

Before Editing the Script

After adding the component to the data flow, double-click the component to open the Script Transformation Editor and follow the steps below:

1. Configure all the input and output columns in the Inputs and Outputs tab. Be sure to set the proper data type for each output, which can be found under the Data Type Properties in the right-hand column.
2. Add any SSIS package variables in the ReadOnlyVariables or ReadWriteVariables lists on the Script tab. Note that read/write variables can only be set in the PostExecute method in the script (see the example script below).
3. Select the language you wish to code with: either C# or Visual Basic.

Now you are ready to begin editing the script. Click the Edit Script button. The Example Script section below shows a typical stored procedure call.

After Editing the Script

After editing the code check if there are any errors in the Error List window and resolve them as needed. The script component is now ready to use.

Example Script

The example below shows how to use a script component to call the SearchSuppliers stored procedure. You will need to add a reference to the TIBCO.SSIS2019.TDV.dll, which can be found in the lib subfolder of the installation folder.

C#

```
using System.Data.TDVClient;
...
public override void CreateNewOutputRows()
{
  string connectionString =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword";
  using (TDVConnection connection = new
TDVConnection(connectionString)) {
    TDVCommand cmd = new TDVCommand("SearchSuppliers", connection);
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.Add(new TDVParameter("@Country", "US"));
    // Add other parameters as needed ...

    TDVDataReader rdr = cmd.ExecuteReader();
    while (rdr.Read()) {
      Output0Buffer.AddRow();
      //Add output columns as necessary, for example:
      //Output0Buffer.Name = rdr["Name"].ToString();
      //...
      Console.WriteLine();
    }
  }
}

public override void PostExecute()
{
  //If you want to set any package variables, it must be done in
this function
  //You will need to have already added these ReadWriteVariables in
the Script Editor
  //For example:
  Variables.Success = true;
}
```

VB.NET

```
Imports System.Data.TDVClient
...
Public Overrides Sub CreateNewOutputRows()
```

```
  Dim connectionString As String =
"Host=myHost;Domain=myDomain;DataSource=myDataSource;User=myUser;P
assword=myPassword"
  Using connection As New TDVConnection(connectionString)
    Dim cmd As New TDVCommand("SearchSuppliers", connection)
    cmd.CommandType = CommandType.StoredProcedure
    cmd.Parameters.Add(New TDVParameter("@Country", "US"))
    ' Add other parameters as needed ...

    Dim rdr As TDVDataReader = cmd.ExecuteReader()
    While rdr.Read()
      Output0Buffer.AddRow()
      'Add output columns as necessary, for example:
      'Output0Buffer.Name = rdr["Name"].ToString()
      '...
      Console.WriteLine()
    End While
  End Using
End Sub

Public Overrides Sub CreateNewOutputRows()
  'If you want to set any package variables, it must be done in this
function
  'You will need to have already added these ReadWriteVariables in
the Script Editor
  'For example:
  Variables.Success = True
End Sub
```

Connection Properties

The connection properties are the various options that can be used to establish a connection. This section provides a complete list of the options you can configure in the connection settings for this provider. Click the links for further details.

For more information on establishing a connection, see Establishing a Connection.

Authentication

Property	Description
Host	The name of the server running TDV Server.
Port	The port of the TDV server.
Domain	The TDV domain to which the DataSource belongs.
DataSource	The name of the TDV data source.
User	The username provided for authentication with TDV Server.
Password	The user's password.
Encrypt	Specifies whether to encrypt the connection using SSL.
SSO	The single-sign-on (SSO) type to use to authenticate.
UserTokens	Authentication values that can be packaged for delivery.

Kerberos

Property	Description
KerberosKDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
KerberosRealm	The Kerberos Realm used to authenticate the user with.
KerberosSPN	The Service Principal Name for the Kerberos Domain Controller.
UsePlatformKerberosAPI	This setting determines if the platform's Kerberos API is used.

SSL

Property	Description
SSLClientCert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).
SSLClientCertType	The type of key store containing the TLS/SSL client certificate.
SSLClientCertPassword	The password for the TLS/SSL client certificate.
SSLClientCertSubject	The subject of the TLS/SSL client certificate.
SSLServerCert	The certificate to be accepted from the server when connecting using TLS/SSL.

Logging

Property	Description
Logfile	A filepath which designates the name and location of the log file.
Verbosity	The verbosity level that determines the amount of detail included in the log file.
LogModules	Core modules to be included in the log file.
MaxLogFileSize	A string specifying the maximum size in bytes for a log file (for example, 10 MB).
MaxLogFileCount	A string specifying the maximum file count of log files.

Schema

Property	Description
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.
BrowsableSchemas	This property restricts the schemas reported to a subset of the available schemas. For example, BrowsableSchemas=SchemaA,SchemaB,SchemaC.
Tables	This property restricts the tables reported to a subset of the available tables. For example, Tables=TableA,TableB,TableC.
Views	Restricts the views reported to a subset of the available tables. For example, Views=ViewA,ViewB,ViewC.

Miscellaneous

Property	Description
Alternate Security Credentials	Uses the URL to set an alternate security credentials value for client authorization when using TDV with a client restricted license.
Case Sensitive	Specifies case sensitivity in the request values.
Catalog	The name of the catalog to use.
Commit Failure	Specifies the behavior if a commit fails.
Commit Interrupt	Specifies the behavior if a commit is interrupted.
Compensate	The correcting behavior.
Connect Timeout	The time-out for initial connection, in seconds.
Default Catalog	The default catalog for a specified connection.
Default Schema	The default schema for a specified connection.
Enable Flood	Values are true or false. Default value is true. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.
Enable Reconnect On Error	Specifies cluster reconnection behavior.

Property	Description
Fetch Bytes	The maximum number of rows to fetch for a batch based on batch size, in bytes.
Fetch Rows	Maximum number of rows to fetch for a batch.
Ignore Trailing Spaces	Specifies whether to ignore trailing spaces at the end of values.
Locale	Value that defines the user's language and country.
Max Rows	A path to the directory that contains the schema files defining tables, views, and stored procedures.
Max Rows	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
No Metadata	Blocks return of result-set metadata during query execution.
Optimization Prepare	Specifies whether to optimize prepare requests sent to TDV.
Other	These hidden properties are used only in specific use cases.
Param Mode	Controls the behavior of OUT parameters for stored procedures.
Query Passthrough	This option passes the query to the TDV server as is.
Readonly	You can use this property to enforce read-only access to TDV from the provider.
Register Output Cursors	Specifies how to handle output cursors.
Request Timeout	The time-out for query commands and other requests, in seconds.
Session Timeout	Session inactivity time-out, in seconds.
Session Token	Uses the URL to set a session token value for client authorization when using TDV with a client restricted license.
Strip Duplicates	Values are true or false. Default value is true. If true, the server will detect duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire.

Property	Description
Strip Trailing Zeros	Determines whether decimal result values are to be returned with trailing zeroes removed.
Trace Folder	The absolute directory to save the trace file.
Trace Level	The level of information to log.
Validate Remote Cert	The username provided for authentication with TDV Server.

Other

Property	Description
EnableFastExec	Specifies whether to enable fast execution of queries.

Authentication

This section provides a complete list of the Authentication properties you can configure in the connection settings for this provider.

Property	Description
Host	The name of the server running TDV Server.
Port	The port of the TDV server.
Domain	The TDV domain to which the DataSource belongs.
DataSource	The name of the TDV data source.
User	The username provided for authentication with TDV Server.
Password	The user's password.
Encrypt	Specifies whether to encrypt the connection using SSL.
SSO	The single-sign-on (SSO) type to use to authenticate.
UserTokens	Authentication values that can be packaged for delivery.

Host

The name of the server running TDV Server.

Data Type

string

Default Value

""

Remarks

This property should be set to the name or network address of the computer running TDV Server.

Port

The port of the TDV server.

Data Type

int

Default Value

9401

Remarks

Set this to the base (plaintext) client port configured on the server.

When Encrypt is enabled, the component will adjust the port accordingly.

Domain

The TDV domain to which the DataSource belongs.

Data Type

string

Default Value

""

Remarks

The TDV domain to which the DataSource belongs.

Typically the domain is 'composite' for installations with locally defined users.

DataSource

The name of the TDV data source.

Data Type

string

Default Value

""

Remarks

Data source refers to the TDV database name published in the Data Services node.

User

The username provided for authentication with TDV Server.

Data Type

string

Default Value

""

Remarks

The username provided for authentication with TDV Server.

Password

The user's password.

Data Type

string

Default Value

""

Remarks

The password provided for authentication with the TDV Server.

Encrypt

Specifies whether to encrypt the connection using SSL.

Data Type

bool

Default Value

false

Remarks

When set to true, automatically passes messages to the SSL port for processing with the TDV SSL Certificate.

SSO

The single-sign-on (SSO) type to use to authenticate.

Data Type

string

Default Value

"Disable"

Remarks

The single-sign-on (SSO) type to use to authenticate. Valid values are: Disable, Kerberos, and NTLM.

Valid on Windows platform only.

Default is "Disable" which forces the client to provide a user and password to authenticate.

UserTokens

Authentication values that can be packaged for delivery.

Data Type

string

Default Value

""

Remarks

Authentication values that can be packaged for delivery.

The URL can pass the user_tokens property to the server at the init command, in the form: " user_tokens=(" NAME "=" VALUE ("," NAME "=" VALUE)* ")"

Kerberos

This section provides a complete list of the Kerberos properties you can configure in the connection settings for this provider.

Property	Description
KerberosKDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
KerberosRealm	The Kerberos Realm used to authenticate the user with.
KerberosSPN	The Service Principal Name for the Kerberos Domain Controller.

Property	Description
UsePlatformKerberosAPI	This setting determines if the platform's Kerberos API is used.

KerberosKDC

The Kerberos Key Distribution Center (KDC) service used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using Windows Authentication. The component will request session tickets and temporary session keys from the Kerberos Key Distribution Center (KDC) service. The Kerberos Key Distribution Center (KDC) service is conventionally colocated with the domain controller. If Kerberos KDC is not specified the component will attempt to detect these properties automatically from the following locations:

Java System Properties: Kerberos settings can be configured in Java using the config file krb5.conf, or using the system properties java.security.krb5.realm and java.security.krb5.kdc. The component will use the system settings if KerberosRealm and KerberosKDC are not explicitly set.

Domain Name and Host: The component will infer the Kerberos Realm and Kerberos KDC from the configured domain name and host as a last resort.

Note: Windows authentication is supported in JRE 1.6 and above only.

KerberosRealm

The Kerberos Realm used to authenticate the user with.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using SPNEGO or Windows Authentication. The Kerberos Realm is used to authenticate the user with the Kerberos Key Distribution Service (KDC). The Kerberos Realm can be configured by an administrator to be any string, but conventionally it is based on the domain name. If Kerberos Realm is not specified the component will attempt to detect these properties automatically from the following locations:

Java System Properties: Kerberos settings can be configured in Java using a config file (krb5.conf) or using the system properties java.security.krb5.realm and java.security.krb5.kdc. The component will use the system settings if KerberosRealm and KerberosKDC are not explicitly set.

Domain Name and Host: The component will infer the Kerberos Realm and Kerberos KDC from the user-configured domain name and host as a last resort. This might work in some Windows environments.

Note: Kerberos-based authentication is supported in JRE 1.6 and above only.

KerberosSPN

The Service Principal Name for the Kerberos Domain Controller.

Data Type

string

Default Value

""

Remarks

If the Service Principal Name on the Kerberos Domain Controller is not the same as the URL that you are authenticating to, set the Service Principal Name here.

UsePlatformKerberosAPI

This setting determines if the platform's Kerberos API is used.

Data Type

bool

Default Value

false

Remarks

This setting determines if the platform's Kerberos API is used. By default no platform APIs are relied on to perform Kerberos authentication. Use of the platform API may be enabled by setting this to True. The default value is False.

Note: This functionality is only available on Windows.

SSL

This section provides a complete list of the SSL properties you can configure in the connection settings for this provider.

Property	Description
SSLClientCert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).
SSLClientCertType	The type of key store containing the TLS/SSL client certificate.
SSLClientCertPassword	The password for the TLS/SSL client certificate.
SSLClientCertSubject	The subject of the TLS/SSL client certificate.
SSLServerCert	The certificate to be accepted from the server when connecting using TLS/SSL.

SSLClientCert

The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).

Data Type

string

Default Value

""

Remarks

The name of the certificate store for the client certificate.

The SSLClientCertType field specifies the type of the certificate store specified by SSLClientCert. If the store is password protected, specify the password in SSLClientCertPassword.

SSLClientCert is used in conjunction with the SSLClientCertSubject field in order to specify client certificates. If SSLClientCert has a value, and SSLClientCertSubject is set, a search for a certificate is initiated. See SSLClientCertSubject for more information.

Designations of certificate stores are platform-dependent.

The following are designations of the most common User and Machine certificate stores in Windows:

MY	A certificate store holding personal certificates with their associated private keys.
CA	Certifying authority certificates.
ROOT	Root certificates.
SPC	Software publisher certificates.

In Java, the certificate store normally is a file containing certificates and optional private keys.

When the certificate store type is PFXFile, this property must be set to the name of the file. When the type is PFXBlob, the property must be set to the binary contents of a PFX file (for example, PKCS12 certificate store).

SSLClientCertType

The type of key store containing the TLS/SSL client certificate.

Data Type

string

Default Value

"USER"

Remarks

This property can take one of the following values:

USER - default	For Windows, this specifies that the certificate store is a certificate store owned by the current user. Note that this store type is not available in Java.
MACHINE	For Windows, this specifies that the certificate store is a machine store. Note that this store type is not available in Java.
PFXFILE	The certificate store is the name of a PFX (PKCS12) file containing certificates.
PFXBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in PFX (PKCS12) format.
JKSFILE	The certificate store is the name of a Java key store (JKS) file containing certificates. Note that this store type is only available in Java.
JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in JKS format. Note that this store type is only available in Java.
PEMKEY_FILE	The certificate store is the name of a PEM-encoded file that contains a private key and an optional certificate.
PEMKEY_BLOB	The certificate store is a string (base64-encoded) that contains a private key and an optional certificate.
PUBLIC_KEY_FILE	The certificate store is the name of a file that contains a PEM- or DER-encoded public key certificate.
PUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains a PEM- or DER-encoded public key certificate.
SSHPUBLIC_KEY_FILE	The certificate store is the name of a file that contains an SSH-style public key.

SSHPUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains an SSH-style public key.
P7BFILE	The certificate store is the name of a PKCS7 file containing certificates.
PPKFILE	The certificate store is the name of a file that contains a PuTTY Private Key (PPK).
XMLFILE	The certificate store is the name of a file that contains a certificate in XML format.
XMLBLOB	The certificate store is a string that contains a certificate in XML format.

SSLClientCertPassword

The password for the TLS/SSL client certificate.

Data Type

string

Default Value

""

Remarks

If the certificate store is of a type that requires a password, this property is used to specify that password to open the certificate store.

SSLClientCertSubject

The subject of the TLS/SSL client certificate.

Data Type

string

Default Value

"*"

Remarks

When loading a certificate the subject is used to locate the certificate in the store.

If an exact match is not found, the store is searched for subjects containing the value of the property. If a match is still not found, the property is set to an empty string, and no certificate is selected.

The special value "*" picks the first certificate in the certificate store.

The certificate subject is a comma separated list of distinguished name fields and values. For example, "CN=www.server.com, OU=test, C=US, E=support@company.com". The common fields and their meanings are shown below.

Field	Meaning
CN	Common Name. This is commonly a host name like www.server.com.
O	Organization
OU	Organizational Unit
L	Locality
S	State
C	Country
E	Email Address

If a field value contains a comma, it must be quoted.

SSLServerCert

The certificate to be accepted from the server when connecting using TLS/SSL.

Data Type

string

Default Value

""

Remarks

If using a TLS/SSL connection, this property can be used to specify the TLS/SSL certificate to be accepted from the server. Any other certificate that is not trusted by the machine is rejected.

This property can take the following forms:

Description	Example
A full PEM Certificate (example shortened for brevity)	-----BEGIN CERTIFICATE----- MIIChTCCAe4CAQAwDQYJKoZIhv......Qw== -----END CERTIFICATE-----
A path to a local file containing the certificate	C:\cert.cer
The public key (example shortened for brevity)	-----BEGIN RSA PUBLIC KEY----- MIGfMA0GCSq......AQAB -----END RSA PUBLIC KEY-----
The MD5 Thumbprint (hex values can also be either space or colon separated)	ecadbdda5a1529c58a1e9e09828d70e4
The SHA1 Thumbprint (hex values can also be either space or colon separated)	34a929226ae0819f2ec14b4a3d904f801cbb150d

If not specified, any certificate trusted by the machine is accepted.

Use '*' to signify to accept all certificates. Note that this is not recommended due to security concerns.

Logging

This section provides a complete list of the Logging properties you can configure in the connection settings for this provider.

Property	Description
Logfile	A filepath which designates the name and location of the log file.
Verbosity	The verbosity level that determines the amount of detail included in the log file.
LogModules	Core modules to be included in the log file.
MaxLogFileSize	A string specifying the maximum size in bytes for a log file (for example, 10 MB).
MaxLogFileCount	A string specifying the maximum file count of log files.

Logfile

A filepath which designates the name and location of the log file.

Data Type

string

Default Value

""

Remarks

Once this property is set, the component will populate the log file as it carries out various tasks, such as when authentication is performed or queries are executed. If the specified file doesn't already exist, it will be created.

Connection strings and version information are also logged, though connection properties containing sensitive information are masked automatically.

If a relative filepath is supplied, the location of the log file will be resolved based on the path found in the Location connection property.

For more control over what is written to the log file, you can adjust the Verbosity property.

Log contents are categorized into several modules. You can show/hide individual modules using the LogModules property.

To edit the maximum size of a single logfile before a new one is created, see MaxLogFileSize.

If you would like to place a cap on the number of logfiles generated, use MaxLogFileCount.

Verbosity

The verbosity level that determines the amount of detail included in the log file.

Data Type

string

Default Value

"1"

Remarks

The verbosity level determines the amount of detail that the component reports to the Logfile. Verbosity levels from 1 to 5 are supported. These are detailed in the Logging page.

LogModules

Core modules to be included in the log file.

Data Type

string

Default Value

""

Remarks

Only the modules specified (separated by ';') will be included in the log file. By default all modules are included.

See the Logging page for an overview.

MaxLogFileSize

A string specifying the maximum size in bytes for a log file (for example, 10 MB).

Data Type

string

Default Value

"100MB"

Remarks

When the limit is hit, a new log is created in the same folder with the date and time appended to the end. The default limit is 100 MB. Values lower than 100 kB will use 100 kB as the value instead.

Adjust the maximum number of logfiles generated with MaxLogFileCount.

MaxLogFileCount

A string specifying the maximum file count of log files.

Data Type

int

Default Value

-1

Remarks

When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted.

The minimum supported value is 2. A value of 0 or a negative value indicates no limit on the count.

Adjust the maximum size of the logfiles generated with MaxLogFileSize.

Schema

This section provides a complete list of the Schema properties you can configure in the connection settings for this provider.

Property	Description
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.
BrowsableSchemas	This property restricts the schemas reported to a subset of the available schemas. For example, BrowsableSchemas=SchemaA,SchemaB,SchemaC.
Tables	This property restricts the tables reported to a subset of the available tables. For example, Tables=TableA,TableB,TableC.
Views	Restricts the views reported to a subset of the available tables. For example, Views=ViewA,ViewB,ViewC.

Location

A path to the directory that contains the schema files defining tables, views, and stored procedures.

Data Type

string

Default Value

"%APPDATA%\\TIBCO\\TDV Data Provider\\Schema"

Remarks

The path to a directory which contains the schema files for the component (.rsd files for tables and views, .rsb files for stored procedures). The folder location can be a relative path from the location of the executable. The Location property is only needed if you want to customize definitions (for example, change a column name, ignore a column, and so on) or extend the data model with new tables, views, or stored procedures.

If left unspecified, the default location is "%APPDATA%\\TIBCO\\TDV Data Provider\\Schema" with %APPDATA% being set to the user's configuration directory:

Platform	%APPDATA%
Windows	The value of the APPDATA environment variable
Mac	~/.config
Linux	~/.config

BrowsableSchemas

This property restricts the schemas reported to a subset of the available schemas. For example, BrowsableSchemas=SchemaA,SchemaB,SchemaC.

Data Type

string

Default Value

""

Remarks

Listing the schemas from databases can be expensive. Providing a list of schemas in the connection string improves the performance.

Tables

This property restricts the tables reported to a subset of the available tables. For example, Tables=TableA,TableB,TableC.

Data Type

string

Default Value

""

Remarks

Listing the tables from some databases can be expensive. Providing a list of tables in the connection string improves the performance of the component.

This property can also be used as an alternative to automatically listing views if you already know which ones you want to work with and there would otherwise be too many to work with.

Specify the tables you want in a comma-separated list. Each table should be a valid SQL identifier with any special characters escaped using square brackets, double-quotes or backticks. For example, Tables=TableA,[TableB/WithSlash],WithCatalog.WithSchema.`TableC With Space`.

Note that when connecting to a data source with multiple schemas or catalogs, you will need to provide the fully qualified name of the table in this property, as in the last example here, to avoid ambiguity between tables that exist in multiple catalogs or schemas.

Views

Restricts the views reported to a subset of the available tables. For example, Views=ViewA,ViewB,ViewC.

Data Type

string

Default Value

""

Remarks

Listing the views from some databases can be expensive. Providing a list of views in the connection string improves the performance of the component.

This property can also be used as an alternative to automatically listing views if you already know which ones you want to work with and there would otherwise be too many to work with.

Specify the views you want in a comma-separated list. Each view should be a valid SQL identifier with any special characters escaped using square brackets, double-quotes or backticks. For example, Views=ViewA,[ViewB/WithSlash],WithCatalog.WithSchema.`ViewC With Space`.

Note that when connecting to a data source with multiple schemas or catalogs, you will need to provide the fully qualified name of the table in this property, as in the last example here, to avoid ambiguity between tables that exist in multiple catalogs or schemas.

Miscellaneous

This section provides a complete list of the Miscellaneous properties you can configure in the connection settings for this provider.

Alternate Security Credentials	Uses the URL to set an alternate security credentials value for client authorization when using TDV with a client restricted license.
Case Sensitive	Specifies case sensitivity in the request values.
Catalog	The name of the catalog to use.
Commit Failure	Specifies the behavior if a commit fails.
Commit Interrupt	Specifies the behavior if a commit is interrupted.
Compensate	The correcting behavior.
Connect Timeout	The time-out for initial connection, in seconds.
Default Catalog	The default catalog for a specified connection.
Default Schema	The default schema for a specified connection.
Enable Flood	Values are true or false. Default value is true. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.
Enable Reconnect On Error	Specifies cluster reconnection behavior.
Fetch Bytes	The maximum number of rows to fetch for a batch based on batch size, in bytes.
Fetch Rows	Maximum number of rows to fetch for a batch.

Ignore Trailing Spaces	Specifies whether to ignore trailing spaces at the end of values.
Locale	Value that defines the user's language and country.
Max Rows	A path to the directory that contains the schema files defining tables, views, and stored procedures.
Max Rows	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
No Metadata	Blocks return of result-set metadata during query execution.
Optimization Prepare	Specifies whether to optimize prepare requests sent to TDV.
Other	These hidden properties are used only in specific use cases.
Param Mode	Controls the behavior of OUT parameters for stored procedures.
Query Passthrough	This option passes the query to the TDV server as is.
Readonly	You can use this property to enforce read-only access to TDV from the provider.
Register Output Cursors	Specifies how to handle output cursors.
Request Timeout	The time-out for query commands and other requests, in seconds.
Session Timeout	Session inactivity time-out, in seconds.
Session Token	Uses the URL to set a session token value for client authorization when using TDV with a client restricted license.
Strip Duplicates	Values are true or false. Default value is true. If true, the server will detect duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire.
Strip Trailing Zeros	Determines whether decimal result values are to be returned with trailing zeroes removed.
Trace Folder	The absolute directory to save the trace file.
Trace Level	The level of information to log.

Validate Remote Cert	The username provided for authentication with TDV Server.
UserTokens	Authentication values that can be packaged for delivery.
Validate Remote Cert	Values are true or false. Default value is false. If true, the client will validate the server's cert.
Validate Remote Hostname	Values are true or false. Default value is false. If true, the client will validate the server's hostname.

Alternate Security Credentials

Uses the URL to set an alternate security credentials value for client authorization when using TDV with a client restricted license.

Data Type

string

Default Value

""

Remarks

Uses the URL to set an alternate security credentials value for client authorization when using TDV with a client restricted license.

Case Sensitive

Specifies case sensitivity in the request values.

Data Type

bool

Default Value

false

Remarks

Specifies case sensitivity in the request values. By default (false), requests are not case-sensitive.

Catalog

The name of the catalog to use.

Data Type

string

Default Value

""

Remarks

This field allows you to limit the Catalog to the one explicitly specified. If not set, the component will retrieve the available catalogs from the TDV server.

Commit Failure

Specifies the behavior if a commit fails.

Data Type

string

Default Value

""

Remarks

Specifies the behavior if a commit fails. Possible values are: rollback or bestEffort.

Commit Interrupt

Specifies the behavior if a commit is interrupted.

Data Type

string

Default Value

""

Remarks

Specifies the behavior if a commit is interrupted. Possible values are: ignore, log, fail.

Compensate

The correcting behavior.

Data Type

string

Default Value

"disabled"

Remarks

The correcting behavior, possible values are: disabled or enabled.

Connect Timeout

The time-out for initial connection, in seconds.

Data Type

int

Default Value

0

Remarks

This property was added for AWS Data Pipeline compatibility. This tool appends this connection property to the URL and by adding this as a hidden property, we avoid the validation error. Has no functional effect in the driver.

The time-out for initial connection, in seconds. Use 0 (zero) for infinite time-out.

Default Catalog

The default catalog for a specified connection.

Data Type

string

Default Value

""

Remarks

The default catalog for a specified connection.

Default Schema

The default schema for a specified connection.

Data Type

string

Default Value

""

Remarks

The default schema for a specified connection.

Enable Flood

Values are true or false. Default value is true. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.

Data Type

bool

Default Value

true

Remarks

Values are true or false. Default value is true. If true, the server will constantly send data, filling the network buffer.Useful for larger result sets.

Enable Reconnect On Error

Specifies cluster reconnection behavior.

Data Type

bool

Default Value

false

Remarks

Specifies cluster reconnection behavior.

Fetch Bytes

The maximum number of rows to fetch for a batch based on batch size, in bytes.

Data Type

int

Default Value

131072

Remarks

The maximum number of rows to fetch for a batch based on batch size, in bytes.

Setting FetchBytes to a very large number can cause an Out Of Memory error in the server. The value set for FetchBytes affects the memory used on the client and the TDV server, so the value should be set based on the heap size configured.

Fetch Rows

Maximum number of rows to fetch for a batch.

Data Type

int

Default Value

500

Remarks

Maximum number of rows to fetch for a batch. Set to 0 (zero) to return an unlimited number of rows.

Ignore Trailing Spaces

Specifies whether to ignore trailing spaces at the end of values.

Data Type

bool

Default Value

false

Remarks

Specifies whether to ignore trailing spaces at the end of values.

Locale

Value that defines the user's language and country.

Data Type

string

Default Value

""

Remarks

Value that defines the user's language and country.

Max Rows

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

Data Type

string

Default Value

"-1"

Remarks

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

No Metadata

Blocks return of result-set metadata during query execution.

Data Type

bool

Default Value

false

Remarks

Blocks return of result-set metadata during query execution.

Optimization Prepare

Specifies whether to optimize prepare requests sent to TDV.

Data Type

bool

Default Value

true

Remarks

When set to "True" (default), the component will submit the query in a single request to TDV.

When set to "False", the component will submit an initial prepare request to TDV.

Other

These hidden properties are used only in specific use cases.

Data Type

string

Default Value

""

Remarks

The properties listed below are available for specific use cases. Normal driver use cases and functionality should not require these properties.

Specify multiple properties in a semicolon-separated list.

Integration and Formatting

OutputKey	Setting this to True will cause the error output to include new output columns containing the primary key values for newly inserted records.
DefaultColumnSize	Sets the default length of string fields when the data source does not provide column length in the metadata. The default value is 2000. Setting this value higher than 4000 will set the SSIS data type to DT_NTEXT allowing for arbitrary length fields.
ConvertDateTimeToGMT	Determines whether to convert date-time values to GMT, instead of the local time of the machine.
RecordToFile=filename	Records the underlying socket data transfer to the specified file.

Param Mode

Controls the behavior of OUT parameters for stored procedures.

Data Type

string

Default Value

"normal"

Remarks

Controls the behavior of OUT parameters for stored procedures.

Valid values are:

normal	Report OUT parameters in procedure metadata as OUT parameters.
return	Report OUT parameters as return values.
omit	Omit OUT parameters from metadata.
omitCursors	Omit output cursors from metadata.

Query Passthrough

This option passes the query to the TDV server as is.

Data Type

bool

Default Value

false

Remarks

When this is set, queries are passed through directly to TDV.

Readonly

You can use this property to enforce read-only access to TDV from the provider.

Data Type

bool

Default Value

false

Remarks

If this property is set to true, the component will allow only SELECT queries. INSERT, UPDATE, DELETE, and stored procedure queries will cause an error to be thrown.

Register Output Cursors

Specifies how to handle output cursors.

Data Type

bool

Default Value

false

Remarks

Specifies how to handle output cursors.

Valid values are:

true	Bind or register output cursors as output parameters.
false	Do not bind or register output cursors as output parameters; instead, use SQLMoreResults or Statement.getMoreResults() to access the cursors.

Request Timeout

The time-out for query commands and other requests, in seconds.

Data Type

int

Default Value

0

Remarks

The time-out for query commands and other requests, in seconds.

Session Timeout

Session inactivity time-out, in seconds.

Data Type

int

Default Value

0

Remarks

Session inactivity time-out, in seconds. Set to 0 (zero) for infinite time-out.

Session Token

Uses the URL to set a session token value for client authorization when using TDV with a client restricted license.

Data Type

string

Default Value

""

Remarks

Uses the URL to set a session token value for client authorization when using TDV with a client restricted license.

Strip Duplicates

Values are true or false. Default value is true. If true, the server will detect duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire.

Data Type

bool

Default Value

true

Remarks

Values are true or false. Default value is true. If true, the server will detect duplicate CHAR/VARCHAR columns in subsequent rows, and will not re-transmit the data across the wire.

Strip Trailing Zeros

Determines whether decimal result values are to be returned with trailing zeroes removed.

Data Type

bool

Default Value

false

Remarks

Determines whether decimal result values are to be returned with trailing zeroes removed.

Trace Folder

The absolute directory to save the trace file.

Data Type

string

Default Value

""

Remarks

The absolute directory to save the trace file.

Trace Level

The level of information to log.

Data Type

string

Default Value

"error"

Remarks

The level of information to log. Valid values are: off, fatal, error (default), warn, info, debug, and all.

Validate Remote Cert

Values are true or false. Default value is false. If true, the client will validate the server's cert.

Data Type

bool

Default Value

false

Remarks

Values are true or false. Default value is false. If true, the client will validate the server's cert.

Validate Remote Hostname

Values are true or false. Default value is false. If true, the client will validate the server's hostname.

Data Type

bool

Default Value

false

Remarks

Values are true or false. Default value is false. If true, the client will validate the server's hostname.

Other

This section provides a complete list of the Other properties you can configure in the connection settings for this provider.

EnableFastExec

Specifies whether to enable fast execution of queries.

Data Type

bool

Default Value

false

Remarks

Values are true or false (default).

Results are processed and returned immediately (instead of round trip) when a query is submitted, potentially improving performance of low latency queries.

TIBCO Product Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join the TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the TIBCO Data Virtualization page.

- Users

  TDV Getting Started Guide

  TDV User Guide

  TDV Web UI User Guide

  TDV Client Interfaces Guide

  TDV Tutorial Guide

  TDV Northbay Example

- Administration

  TDV Installation and Upgrade Guide

  TDV Administration Guide

  TDV Active Cluster Guide

  TDV Security Features Guide

- Data Sources

  TDV Adapter Guides

  TDV Data Source Toolkit Guide (Formerly Extensibility Guide)

- References

  TDV Reference Guide

  TDV Application Programming Interface Guide

- Other

  TDV Business Directory Guide

  TDV Discovery Guide
- *TIBCO TDV and Business Directory Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to TIBCO Support website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, visit TIBCO Community.

Legal and Third-Party Notices