

# **TIBCO® Data Virtualization**

## **Client Interfaces Guide**

*Version 8.1*

*Last Updated: March 25, 2019*

## Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENTATION IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENTATION IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO and the TIBCO logo are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries

TIBCO, Two-Second Advantage, TIBCO Spotfire, TIBCO ActiveSpaces, TIBCO Spotfire Developer, TIBCO EMS, TIBCO Spotfire Automation Services, TIBCO Enterprise Runtime for R, TIBCO Spotfire Server, TIBCO Spotfire Web Player, TIBCO Spotfire Statistics Services, S-PLUS, and TIBCO Spotfire S+ are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENTATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENTATION. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENTATION AT ANY TIME.

THE CONTENTS OF THIS DOCUMENTATION MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Copyright © 2004-2019 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information



# Contents

<b>Preface</b>	<b>11</b>
Product-Specific Documentation	11
How to Access TIBCO Documentation	12
How to Contact TIBCO Support	12
How to Join TIBCO Community	12
<b>Introduction to Accessing Your Data through Client Interfaces</b>	<b>15</b>
About Client Interface Connections	15
Connecting Client Applications to TDV Resources	16
TDV Port Settings for Client Connections to TDV	17
TDV Data Retrieval Tuning for Client Connections to TDV	19
<b>Connecting to TDV Server through JDBC</b>	<b>21</b>
Installing JDBC Drivers	21
Updating the JDBC Driver	22
Setting the Java CLASSPATH for the JDBC Driver	22
Setting Pass-Through Credentials for JDBC Clients	23
Connecting to TDV Server through TIBCO Spotfire	26
Connecting to TDV Server through Squirrel	27
Defining a JDBC Client using a Connection URL	28
JDBC Driver Connection URL Properties	29
Examples	34
Example Java JDBC Client Application Code	34
Example of Calling Procedures	37
Examples of Accessing Data through JDBC	38
Examples of Accessing Data through JDBC Using Statements	39
Examples of Accessing Data through JDBC Using Prepared Statements	43
Tips From an Expert on Duplicate Schema Names in a Catalog	49
Unsupported JDBC Methods	49
<b>Connecting to TDV Server through ODBC</b>	<b>59</b>
ODBC Driver Requirements	60
Installing the ODBC Driver	60
Installing the ODBC Client Driver on Windows	60
Uninstalling the ODBC Client Driver on Windows	62

Updating the ODBC Driver .....	62
Preparing TDV Data Services for ODBC Client Connections .....	62
Configuring Each Windows System Data Source Name .....	63
Adding a New System DSN .....	64
Override the Configured Settings .....	67
Defining an ODBC Client using a Connection URL .....	67
ODBC Driver Connection URL Properties .....	69
Connecting Cognos to TDV Using ODBC .....	74
Connecting Oracle Database Gateway to TDV Using ODBC .....	75
Connecting MicroStrategy to TDV Using ODBC .....	76
Connecting Tableau to TDV Using ODBC .....	79
Connecting PowerBI to TDV Using ODBC .....	80
Examples Using ODBC to Connect to TDV Server .....	81
PERL Code Sample for Connecting to TDV Server .....	81
C++ Example using the Connection URL (DSN-less connection) .....	82
C++ UNIX Code Sample for Connecting to TDV Server .....	84
VBA Code Sample for Connecting to TDV Server .....	87
<b>TIBCO Power BI Data Connector for TDV .....</b>	<b>89</b>
Overview .....	89
Getting Started .....	89
Installing the Connector .....	89
Install the Connector .....	89
Creating the Data Source Name .....	89
Editing the DSN Configuration .....	90
Getting Data .....	90
Connect to the Data .....	90
Load or Edit the Data .....	91
Advanced Settings .....	91
Using the Power BI Connector in Multi-User Environments .....	91
Using the Connector .....	92
Querying Data .....	92
Using DirectQuery .....	92
Using Data Import .....	92
Advanced Connection Properties (optional) .....	93
Advanced Options (optional) .....	93
Visualizing Data .....	93
Creating and Working with Data Visualizations .....	93
Highlighting and Filtering Data .....	93
Creating Real-Time Visualizations .....	94

Connection String Options . . . . .	94
Remarks . . . . .	94
Connection String Options . . . . .	94
Case Sensitive . . . . .	97
Catalog . . . . .	98
Commit Failure . . . . .	98
Commit Interrupt . . . . .	98
Compensate . . . . .	99
Connect Timeout . . . . .	99
Data Source . . . . .	99
Default Catalog . . . . .	100
Default Schema . . . . .	100
Direct Query Limit . . . . .	100
Domain . . . . .	101
Enable Failover . . . . .	101
Enable Fast Exec . . . . .	102
Enable Reconnect On Error . . . . .	102
Encrypt . . . . .	102
Fetch Bytes . . . . .	103
Fetch Rows . . . . .	103
Host . . . . .	104
Ignore Trailing Spaces . . . . .	104
Kerberos KDC . . . . .	104
Kerberos Realm . . . . .	105
Kerberos SPN . . . . .	106
Locale . . . . .	106
Location . . . . .	107
Logfile . . . . .	107
Maximum Column Size . . . . .	107
Max Log File Size . . . . .	108
Max Rows . . . . .	109
No Metadata . . . . .	109
Optimization Prepare . . . . .	109
Other . . . . .	110
Param Mode . . . . .	111
Password . . . . .	111
Port . . . . .	111
Query Passthrough . . . . .	112
Readonly . . . . .	112
Register Output Cursors . . . . .	113
Request Timeout . . . . .	113
Session Timeout . . . . .	114
SSL Client Cert . . . . .	114
SSL Client Cert Password . . . . .	115
SSL Client Cert Subject . . . . .	115

SSL Client Cert Type .....	116
SSL Server Cert .....	118
SSO .....	119
Strip Trailing Zeros .....	119
Tables .....	120
Trace Folder .....	120
Trace Level .....	121
User .....	121
User Tokens .....	121
Verbosity .....	122
Views .....	123
<b>Connecting to TDV Server through Web Interfaces .....</b>	<b>125</b>
Connecting to TDV Server through SOAP .....	125
SOAP Message Compression .....	126
SOAP Message Optimization .....	127
Connecting to TDV Server through REST .....	127
Connecting to TDV Server through OData .....	128
<b>Connecting to TDV Server through ADO.NET .....</b>	<b>131</b>
Setting Up the ADO.NET Driver .....	131
Client-Side ADO.NET Driver Support .....	131
Installing the ADO.NET Driver .....	132
Uninstalling and Repairing ADO.NET .....	132
Updating the ADO.NET Driver .....	133
Configure an ADO.NET Connection to a Client Restricted Server .....	133
Adding and Configuring a Connection to TDV in Visual Studio .....	134
Modifying or Deleting a Connection .....	141
Working with the Server Explorer .....	141
Working with the Visual ToolBox Items .....	143
Sample Code for Testing of an ADO.NET Driver .....	144
Create a CompositeConnection Object .....	145
Create a CompositeCommand Object .....	146
Using a SQL Statement to Create the CompositeCommand Object .....	147
Using the conn.CreateCommand Method to Create the CompositeCommand Object .....	147
Using CompositeCommand to Create the CompositeCommand Object .....	147
Select Data from a TDV Published Resource .....	147
Select Data from a TDV Published Resource on the Server .....	148
Getting the Column Type .....	149
Getting Column Metadata .....	150
Using an Update Operation in the Sample Code .....	151



- About Using Parameters ..... 152
- About ADO.NET Placeholders ..... 153
- Invoking a Stored Procedure Example ..... 154
- Using CompositeCommandBuilder. .... 157
- Example with Special Data Types ..... 158
- Retrieving Metadata ..... 160
- Retrieving Tables with a Named Schema. .... 161



# Preface

---

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, please visit:

- <https://docs.tibco.com>

## Product-Specific Documentation

The following documents form the TIBCO® Data Virtualization(TDV) documentation set:

- *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.
- TDV Installation and Upgrade Guide
- TDV Administration Guide
- TDV Reference Guide
- TDV User Guide
- TDV Security Features Guide
- TDV Business Directory Guide
- TDV Application Programming Interface Guide
- TDV Tutorial Guide
- TDV Extensibility Guide
- TDV Getting Started Guide
- TDV Client Interfaces Guide
- TDV Adapter Guide
- TDV Discovery Guide
- TDV Active Cluster Guide
- TDV Monitor Guide
- TDV Northbay Example

## How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website mainly in the HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit <https://docs.tibco.com>.

Documentation for TIBCO Data Virtualization is available on <https://docs.tibco.com/products/tibco-data-virtualization-server>.

## How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit <https://www.tibco.com/services/support>.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at <https://support.tibco.com>.
- 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 <https://support.tibco.com>. 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](https://community.tibco.com). For a free registration, go to <https://community.tibco.com>.

### Document Change History

The table below provides the revision history for the *TDV Client Interfaces Guide*.

Version Number	Issue Date	Status	Reason for Change
8.0	July 2017		<ul style="list-style-type: none"><li>• Major release. No substantive changes..</li></ul>





# 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:

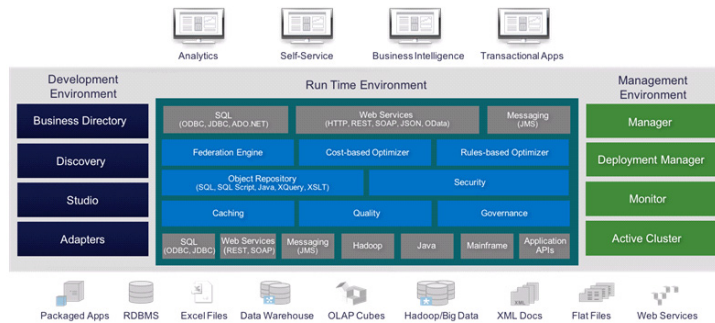
- [About Client Interface Connections, page 15](#)
- [Connecting Client Applications to TDV Resources, page 16](#)
- [TDV Port Settings for Client Connections to TDV, page 17](#)
- [TDV Data Retrieval Tuning for Client Connections to TDV, page 19](#)

## 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.)



5. Configure the connections between TDV and the client applications as described in one of the following sections:

Connector Type	See
JDBC	<a href="#">Connecting to TDV Server through JDBC, page 21</a>
ODBC	<a href="#">Connecting to TDV Server through ODBC, page 59</a>
SOAP	<a href="#">Connecting to TDV Server through SOAP, page 125</a>
REST	<a href="#">Connecting to TDV Server through REST, page 127</a>
OData	<a href="#">Connecting to TDV Server through OData, page 128</a>
ADO.Net	<a href="#">Connecting to TDV Server through ADO.NET, page 131</a>

6. Test your client application access to TDV.

## TDV Port Settings for Client Connections to TDV

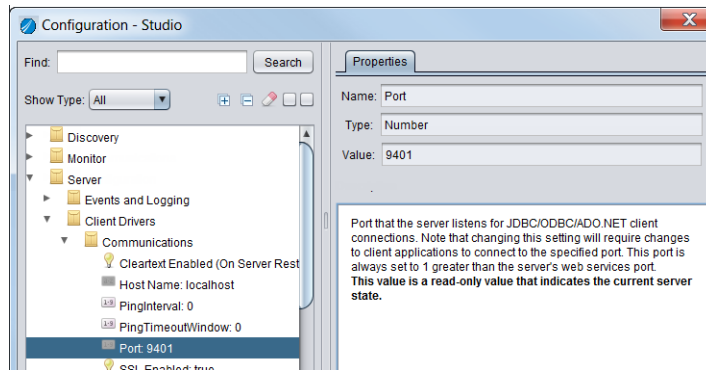
The port settings for clients connecting to TDV using JDBC, ODBC, ADO.NET, and SSL are derived from the HTTP base port setting which is 9400 by default. The default settings for client connections are as follows:

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
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.



### 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	<p>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.</p> <p>If you observe an increase in memory usage this parameter can be used to control the amount of memory used to prefetch data.</p> <p>0: No queuing or prefetch of the results done on the server.</p> <p>1 – 100:</p> <ul style="list-style-type: none"><li>• <b>If prefetch optimization is enabled:</b> 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.</li><li>• <b>If prefetch optimization is disabled:</b> Is always interpreted as 1 and the buffer size is the lesser of fetchBytes or fetchRows.</li></ul>

It is recommended that this feature be turned on.

**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, page 21](#)
- [Updating the JDBC Driver, page 22](#)
- [Setting the Java CLASSPATH for the JDBC Driver, page 22](#)
- [Setting Pass-Through Credentials for JDBC Clients, page 23](#)
- [Connecting to TDV Server through TIBCO Spotfire, page 26](#)
- [Connecting to TDV Server through SquirrelL, page 27](#)
- [Defining a JDBC Client using a Connection URL, page 28](#)
- [Examples, page 34](#)
- [Tips From an Expert on Duplicate Schema Names in a Catalog, page 49](#)
- [Unsupported JDBC Methods, page 49](#)

## 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 21](#).

**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 24](#) 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/dsPassThru",
        "dsUser", "dsPassword");
```

5. If you are using pooled connections, add 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
<a href="#">Example 2: Set Data Source Credentials to Use Pass-through Data Sources, page 40</a>	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.
<a href="#">Example 3: Setting Credentials for Use with Any Pass-through Data Source, page 41</a>	If no credentials have been set on the data source from which data is required, the TDV Server checks to see if the <code>setDataSourceCredentials()</code> method was used to set credentials on the null data source (NULL). If <code>setDataSourceCredentials()</code> was used to set credentials on the null data source, those credentials are used to attempt data retrieval.

Reference	Description
<a href="#">Example 4: Set TDV Server Credentials to Use Pass-through Data Sources, page 42</a>	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 SquirrelL

SquirrelL 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 SquirrelL 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 21](#)
  - [Updating the JDBC Driver, page 22](#)
  - [Setting the Java CLASSPATH for the JDBC Driver, page 22](#)
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 to create a new driver.
6. Type values for the following:

Field	For Example
Name	TDV Server
Example URL	jdbc:compositesw:dbapi@localhost:9401?domain=composite&dataSource=system
Class Name	cs.jdbc.driver.CompositeDriver

7. Select Add.
8. Type the full directory path location to your csjdbc.jar file.
9. Exit to get back the main screen.
10. Select the Alias tab.

11. Click on the blue icon with a plus (+) sign to create a new alias.
12. Type values for the following:

Field	For Example
Name	DEV_TDV_Server<ver>
Driver	TDV Server
URL	jdbc:compositesw:dbapi@lctcvd0250:9401?domain=composite&dataSource=BBMS
User Name	Use and existing account.
Password	

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

13. 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 29](#)

### 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=Composite 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 29](#).

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
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	Correcting behavior, possible values: disabled or enabled.
connectTimeout	Time-out for initial connection, in seconds. Use 0 (zero) for infinite time-out.

JDBC Property Name	Description
convertToLocalTime zone	<p>When set to the default value (false), the client receives <b>TIMESTAMP</b> 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 <b>TIMESTAMP</b>, that value is preserved.</p> <p>When set to true, <b>TIMESTAMP</b> values are converted to the client's time zone (as in some earlier releases of TDV).</p>
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.
enableTDVTimestamp	<p>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:</p> <ul style="list-style-type: none"> <li>• <code>ResultSet.getObject()</code> returns a custom child class of <code>java.sql.Timestamp</code>, with unchanged timestamp.</li> <li>• <code>ResultSet.getTime()</code> returns a <code>java.sql.Time</code> object with unchanged time (hour, minute, second), but the time from epoch is changed.</li> <li>• <code>ResultSet.getString()</code> returns the unchanged timestamp text.</li> <li>• <code>ResultSet.getTimestamp()</code> returns <code>java.sql.Timestamp</code> with changed timestamp regardless of the value of this property.</li> </ul> <p>Notes:</p> <p>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).</p> <p>2) This JDBC property only affects <code>getObject()</code>, <code>getTime()</code>, and <code>getString()</code> of class <code>ResultSet</code>.</p>
enableFastExec	<p>Values are true or false, and the default value is false.</p> <p>Results are processed and returned immediately (instead of a round trip) when a query is submitted, potentially improving performance of low latency queries.</p>

JDBC Property Name	Description
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 <a href="#">“Web Services Security”</a> in the <i>TDV User Guide</i> .
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: <ul style="list-style-type: none"> <li>• normal—Report OUT parameters in procedure metadata as OUT parameters.</li> <li>• return—Report OUT parameters as return values.</li> <li>• omit—Omit OUT parameters from metadata.</li> <li>• omitCursors—Omit output cursors from metadata.</li> </ul>
pingInterval	Maximum time to wait before sending a ping request while waiting for result from TDV, in seconds.

JDBC Property Name	Description
pingTimeoutWindow	<p>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.</p> <p>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.</p>
registerOutputCursors	<ul style="list-style-type: none"> <li>• true—Bind or register output cursors as output parameters.</li> <li>• false—Do not bind or register output cursors as output parameters; instead, use <code>SQLMoreResults</code> or <code>Statement.getMoreResults()</code> to access the cursors. See <a href="#">Example of Calling Procedures, page 37</a>.</li> </ul>
requestTimeout	Time-out for query commands and other requests.
sessionTimeout	Session inactivity time-out, in seconds. Set to zero for infinite time-out.
sessionToken	<p>Uses the JDBC URL to set a session token value for client authorization when using TDV with a client restricted license.</p> <p>Example: <code>&amp;sessionToken=&lt;VALUE&gt;</code></p>
singleLogSize	Maximum log file size to saving to next log file, in M bytes.
stripTrailingZeros	Determines whether decimal result values are to be returned with trailing zeros removed.
traceLevel	<p>Valid values are off, fatal, error (this is the default), debug, warn, info, debug, and all.</p> <p>The valid values for client-side log settings are off, fatal, error (default), warn, info, debug, all, stdout.</p> <p>On UNIX-based platforms, the log file <code>CsOdbcDebug.log</code> is created in the directory specified by the environment variable <code>COMPOSITE_HOME</code>.</p>



JDBC Property Name	Description
unsupportedMode	<p>Valid values are silent, warn, or fail. The default value is fail.</p> <p>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 <a href="#">Unsupported JDBC Methods, page 49</a>.</p>
user_tokens	<p>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:</p> <pre>"user_tokens=(" NAME "=" VALUE ( "," NAME "=" VALUE ) * ")"</pre>
validateRemoteCert	<p>Windows platform only. Ignored on UNIX platforms.</p> <p>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.</p> <p>True: The TDV JDBC client initiates the validation handshake, using the TDV certificate a for password encryption. If validation fails, no connection is established.</p> <p>The TDV Server certificate is loaded from the file specified in the Truststore File Location configuration parameter.</p> <p>The Keystore Key Alias is used when it is configured.</p> <p>The TDV JDBC client driver uses the client system's truststore properties to validate the certificate:</p> <ul style="list-style-type: none"> <li>• javax.net.ssl.trustStore</li> <li>• javax.net.ssl.trustStorePassword</li> <li>• javax.net.ssl.trustStoreType</li> </ul> <p>The TDV Server certificate must be added to this client's truststore; otherwise, validation fails.</p> <p>The placeholder TDV certificate does not work after the client system truststore is enabled, unless it is added to the client truststore.</p>

JDBC Property Name	Description
validateRemoteHost name	<p>Windows platform only. Ignored on UNIX platforms.</p> <p>False (default): No host name validation is performed.</p> <p>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.</p> <p>If host name validation fails, the connection is not established.</p>

## Examples

- [Example Java JDBC Client Application Code, page 34](#)
- [Examples of Accessing Data through JDBC, page 38](#)

### 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).clearAllDataSourceCredentials();
    ((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 39](#)
- [Examples of Accessing Data through JDBC Using Prepared Statements, page 43](#)

### 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 39](#)
- [Example 2: Set Data Source Credentials to Use Pass-through Data Sources, page 40](#)
- [Example 3: Setting Credentials for Use with Any Pass-through Data Source, page 41](#)
- [Example 4: Set TDV Server Credentials to Use Pass-through Data Sources, page 42](#)

### 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());
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

    }
    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/dsPassThru", "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 40](#). 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/dsPassThru" "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 44](#)
- [Prepared Statements Examples, page 44](#)

## 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 be 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 44](#)
- [Example 2: INSERT Statement, page 46](#)
- [Example 3: UPDATE Statement, page 47](#)
- [Example 4: DELETE Statement, page 48](#)

### 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=cdbbs";

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 44](#), 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 46](#). 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 46](#). 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)

PreparedStatement (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, page 60](#)
- [Installing the ODBC Driver, page 60](#)
- [Updating the ODBC Driver, page 62](#)
- [Preparing TDV Data Services for ODBC Client Connections, page 62](#)
- [Configuring Each Windows System Data Source Name, page 63](#)
- [Adding a New System DSN, page 64](#)
- [Override the Configured Settings, page 67](#)
- [Defining an ODBC Client using a Connection URL, page 67](#)
- [Connecting Cognos to TDV Using ODBC, page 74](#)
- [Connecting Oracle Database Gateway to TDV Using ODBC, page 75](#)
- [Connecting MicroStrategy to TDV Using ODBC, page 76](#)
- [Connecting Tableau to TDV Using ODBC, page 79](#)
- [Connecting PowerBI to TDV Using ODBC, page 80](#)
- [Examples Using ODBC to Connect to TDV Server, page 81](#)

# 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 60](#)
- [Updating the ODBC Driver, page 62](#)

To remove the ODBC client driver on Windows, see:

- [Uninstalling the ODBC Client Driver on Windows, page 62](#)

## 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. a 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

## 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 17](#).

### 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 17](#) 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.

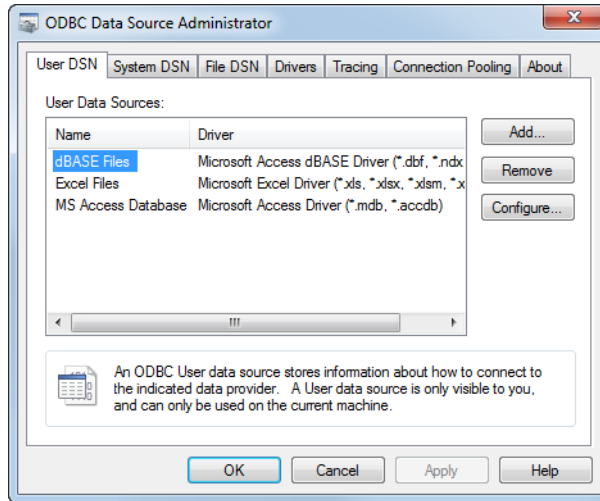
## Adding a New System DSN

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 7, 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.



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).
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	<p>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.</p> <p>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.</p> <p>To retrieve the client user name within the CJP execution context /HOOK running on TDV, call <code>ExecutionEnvironment.getProperty(loggedInUser)</code>.</p> <p>TDV does not use the client's Windows log-in user name for authentication.</p>
Password	Password corresponding to user name.
Domain	Domain to which the user belongs.
Datasource	<p>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.</p> <p>For example, if you have published resources under test (shown as /services/databases/test in the tool tip), type just "test" in this field.</p>

Field	Description
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.

- 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.

- 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	<p>The subject (CN) of Windows private certificate. You can locate it using this procedure:</p> <ol style="list-style-type: none"> <li>Run the certmgr.msc program to open the Windows Certificate Manager.</li> <li>Under Personal/Certificates, double-click the private certificate to open the Certificate dialog.</li> <li>Click the Details tab, then select the Subject field.</li> </ol> <p>Enter the Subject CN value in the SSL Key ID field. This is the only parameter you need to specify for Windows.</p>
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)
SSL CA Path	<p>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)</p> <p>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.</p>

Field	Description
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>;PORT=<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 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:

- [ODBC Driver Connection URL Properties](#), page 69
- [C++ Example using the Connection URL \(DSN-less connection\)](#), page 82

**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";
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 URL properties, see [ODBC Driver Connection URL Properties](#), page 69.

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 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);
```

## ODBC Driver Connection URL Properties

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

ODBC Property	Description
caseSensitive	Specifies case sensitivity in the request values. By default ( <b>false</b> ), requests are not case-sensitive.
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. Possible values are: disabled or enabled.
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.
enableReconnectOnError	Specifies cluster reconnection behavior.

ODBC 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: <b>false</b> .
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: <ul style="list-style-type: none"> <li>• normal—Report OUT parameters in procedure metadata as OUT parameters.</li> <li>• return—Report OUT parameters as return values.</li> <li>• omit—Omit OUT parameters from metadata.</li> <li>• omitCursors—Omit output cursors from metadata.</li> </ul>
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	<p>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.</p> <p>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.</p>
port	TDV Server listening port.
registerOutputCursors	<ul style="list-style-type: none"> <li>• true—Bind or register output cursors as output parameters.</li> <li>• false—Do not bind or register output cursors as output parameters; instead, use SQLMoreResults to access the cursors.</li> </ul>
requestTimeout	Time-out for query commands and other requests
sessionTimeout	Session inactivity timeout, in seconds. Set to zero for infinite timeout.
sessionToken	<p>Uses the URL to set a session token value for client authorization when using TDV with a client restricted license.</p> <p>Example: &amp;sessionToken=&lt;VALUE&gt;</p>
singleLogSize	Maximum log file size to saving to next log file, in M bytes.
spn	<p>Valid on Windows platform only, not useful on UNIX platforms.</p> <p>Kerberos SPN value, only useful if the SSO value equals Kerberos.</p>
sso	<p>Valid on Windows platform only, not useful on UNIX platforms.</p> <p>Single-sign-on type: ""/(Disabled), Kerberos or NTLM.</p> <p>The default value is "", which forces the ODBC client application to provide user and password information to connect.</p>

ODBC Property	Description
sslKeyID	<p>The subject (CN) of the Windows private certificate. You can locate this using this procedure:</p> <ol style="list-style-type: none"> <li>1. Run the certmgr.msc program to open the Windows Certificate Manager.</li> <li>2. Under Personal/Certificates, double-click the private certificate to open the Certificate dialog.</li> <li>3. Click the Details tab, then select the Subject field.</li> </ol> <p>Enter the Subject CN value in the sslKeyID field. This is the only parameter you need to specify for Windows.</p>
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)
sslCAPath	<p>The absolute path of the directory that contains the trusted CA files in PEM format. On the Linux platform, the default value is <code>"/etc/ssl/certs"</code>. The CA PEM file name in the CA path directory must equal the hash value for the CA PEM file name. (Optional)</p> <p>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.</p>
stripTrailingZeros	Determines whether decimal result values are to be returned with trailing zeros removed.
traceFile	Absolute path to the trace file.
traceFolder	Absolute directory to save trace file, the trace file name is <code>"CsOdbcDebug_" + &lt;DSN Name&gt; + ".log"</code> . the default folder is <code>C:\</code> or <code>\$COMPOSITE_HOME</code>



ODBC Property	Description
traceLevel	<p>Valid values are off, fatal, error (this is the default), debug, warn, info, debug, and all.</p> <p>The valid values for client-side log settings are off, fatal, error (default), warn, info, debug, all, stdout.</p> <p>On UNIX-based platforms, the log file CsOdbcDebug.log is created in the directory specified by the environment variable COMPOSITE_HOME.</p>
unsupportedMode	<p>Valid values are silent, warn, or fail. The default value is fail.</p> <p>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.</p>
user_tokens	Authentication values that can be packaged for delivery.
user/uid	Specifies the user name for connections to the data source.
validateRemoteCert	<p>Useful on Windows platform only, it is ignored on UNIX platforms.</p> <p>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.</p> <p>When false (default), no certificate validation is performed prior to the establishment of a connection.</p> <p>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 <i>TDV Administration Guide</i>.</p> <p>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.</p>
validateRemoteHostname	<p>Useful on Windows platform only, it is ignored on UNIX platforms.</p> <p>When true, the ODBC 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.</p> <p>If the host name validation fails, the connection is not established. When false (default), the host name validation is not performed.</p>

# 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 or Solaris	LD_LIBRARY_PATH
AIX	LIBPATH

System Type	Environment Variable
HP-UX	SH_LIB_PATH

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 or Solaris	LD_LIBRARY_PATH
AIX	LIBPATH
HP-UX	SH_LIB_PATH

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 or Solaris	LD_LIBRARY_PATH
AIX	LIBPATH
HP-UX	SH_LIB_PATH

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 URL 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.
21. (Optionally) If you are on Solaris and using the MicroStrategy DB Query tool, you must specify that the ODBC Driver is Non UTF-8.

## 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 TIMESTAMP data type. For example, if your time value is 12:05, your converted TIMESTAMP will be 1900-01-01 12:05.

### 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 60](#).

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 64](#).
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.

### 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 64](#).
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 81](#)
- [C++ Example using the Connection URL \(DSN-less connection\), page 82](#)
- [C++ UNIX Code Sample for Connecting to TDV Server, page 84](#)
- [VBA Code Sample for Connecting to TDV Server, page 87](#)

### 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 URL (DSN-less connection)

The following example shows a small test program written in C++ for Windows that uses this connection URL method. The string “dsn” uses the connection string URL 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)
{
    SQLELEN 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(SQLUIINTEGER)=%d\n",sizeof(SQLUIINTEGER));
    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;validateRemot
eHostname=false;connectTimeout=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,(SQLCHAR*)"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 URL 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(SQLINTEGER)=%d\n",sizeof(SQLINTEGER));
    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;validateRemot
eHostname=false;connectTimeout=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, (SQLCHAR*)"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 setup.
2. Setup prompts you to install Microsoft Visual C++ Redistributable for Visual Studio 2017. 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.

## 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	The correcting behavior.
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.
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**

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.  
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.
JKS BLOB	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----- MIChTCCAe4CAQAwDQYJKoZIhvd.....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-----

Description	Example
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 verbosity, 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: 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, page 125](#)
- [SOAP Message Compression, page 126](#)
- [SOAP Message Optimization, page 127](#)
- [Connecting to TDV Server through REST, page 127](#)
- [Connecting to TDV Server through OData, page 128\](#)

## 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>:<HTTTPs_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.

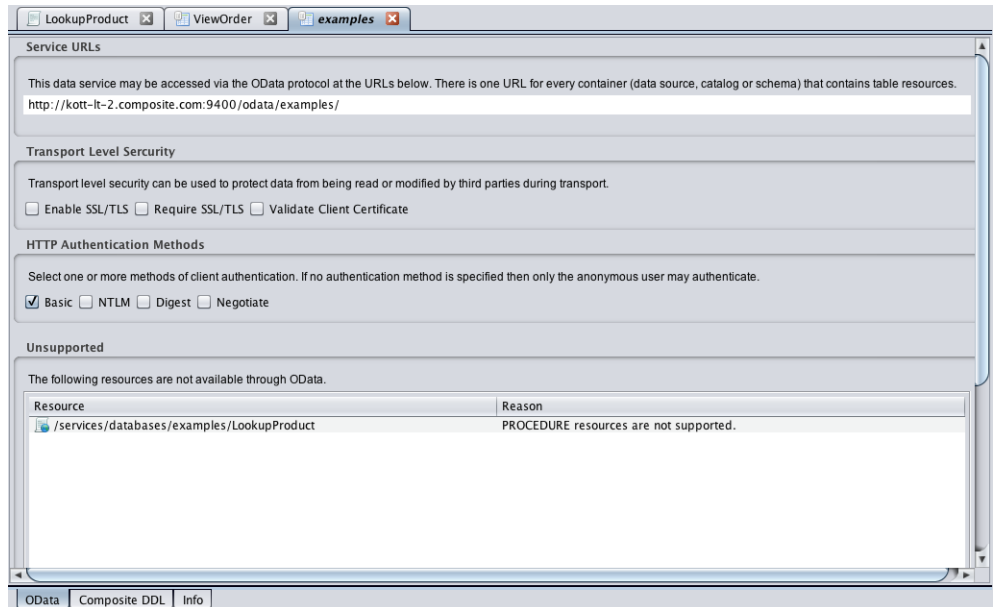
OData with TDV:

- 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.



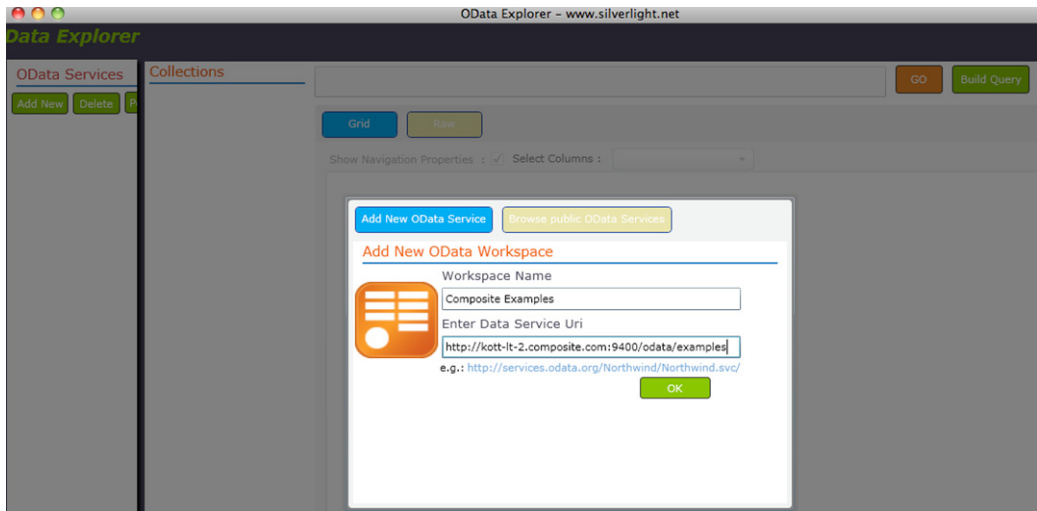
## To define the connection between TDV Server and OData

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



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:



# 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, page 131](#)
- [Configure an ADO.NET Connection to a Client Restricted Server, page 133](#)
- [Adding and Configuring a Connection to TDV in Visual Studio, page 134](#)
- [Modifying or Deleting a Connection, page 141](#)
- [Working with the Server Explorer, page 141](#)
- [Working with the Visual ToolBox Items, page 143](#)
- [Sample Code for Testing of an ADO.NET Driver , page 144](#)

## Setting Up the ADO.NET Driver

This section includes the following:

- [Client-Side ADO.NET Driver Support, page 131](#)
- [Installing the ADO.NET Driver, page 132](#)
- [Uninstalling and Repairing ADO.NET, page 132](#)
- [Updating the ADO.NET Driver, page 133](#)
- [Configure an ADO.NET Connection to a Client Restricted Server, page 133](#)

## 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 2008 R2
- Windows Server 2012 R2
- Windows Server 2016 R2

The TDV ADO.NET driver requires the following pre-installed software:

- .NET Framework 2.0

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, ADONetInstall.msi. Obtain this from the client driver zip file.

### To install the ADO.NET driver

1. Make sure Visual Studio is closed.
2. Double click or open ADONetInstall.msi.
3. The ADO.NET Driver for TDV <version> installer walks you through installation.

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 ADONetInstall.msi installer on a computer that has an ADO.NET driver installed.
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 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;sessionToken=ffa29823a2a8e340f20f15048aeebcdd746387d235abc12;
```

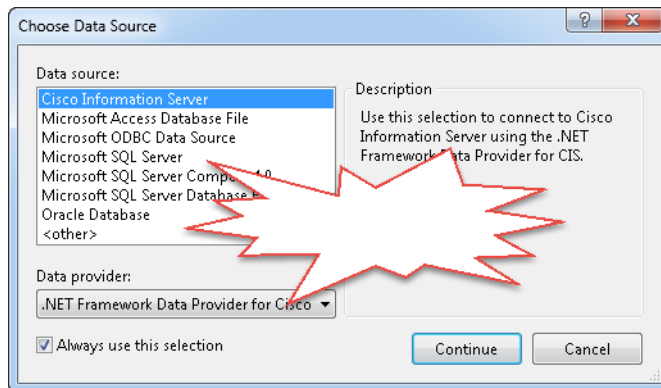
## Adding and Configuring a Connection to TDV in Visual Studio

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 the TDV 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.



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

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.

**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**

Property	Value
<b>Advanced</b>	
ConnectTimeout	
FetchBytes	
FetchRows	
RequestTimeout	
SessionTimeout	
SessionToken	
<b>Cluster</b>	
EnableFailover	false
EnableReconnectOnError	True
PingInterval	
PingTimeoutWindow	
<b>Connection</b>	
Catalog	
DataSource	MyCISDataSource
Domain	composite
Host	localhost
Password	*****
Port	9401
User	admin
<b>Debug</b>	
ErrorLoggingEnabled	
StatusInterval	
TraceFolder	
TraceLevel	off
<b>Pooling</b>	
ConnectionLifetime	0
MaxPoolSize	100
MinPoolSize	0
Pooling	True
<b>Security</b>	
Encrypt	False
Integrated_Authentication	(Disabled)
Kerberos SPN	
KeyStoreFile	
KeyStorePass	
User_Tokens	
ValidateRemoteCert	False
ValidateRemoteHostname	False

**Security**

tracelevel=off;traceperf=off;locale=en-us;encrypt=False;validateremotecert=False

OK Cancel

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

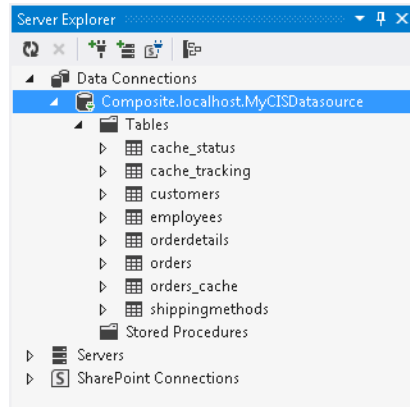
Parameter	Value description
Advanced	
ConnectTimeout	The number of seconds the client waits for a connection to be established or to fail. A value of 0 disables the timeout.
FetchBytes	Maximum number of rows to fetch for a batch based on batch size, in bytes. Setting fetchBytes to a very large number 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	<p>The maximum number of rows fetched from TDV at one time.</p> <p><b>Note:</b> 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.</p>
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	
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.

Parameter	Value description
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.
<b>Connection</b>	Properties in this section are values that were set in <a href="#">Adding and Configuring a Connection to TDV in Visual Studio, page 134</a> . 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.
<b>Debug</b>	
ErrorLoggingEnabled	
SatusInterval	
TraceFolder	
TraceLevel	off or on. Enables or disables debug level logging.
<b>Pooling</b>	
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

Parameter	Value description
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	
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.
KeyStorePass	Used for SSL security. Specifies the password for the KeyStoreFile. The default value is 'changeit'.
User_Tokens	
ValidateRemoteCert	True or False. The default is False.
ValidateRemoteHost name	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.



## 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

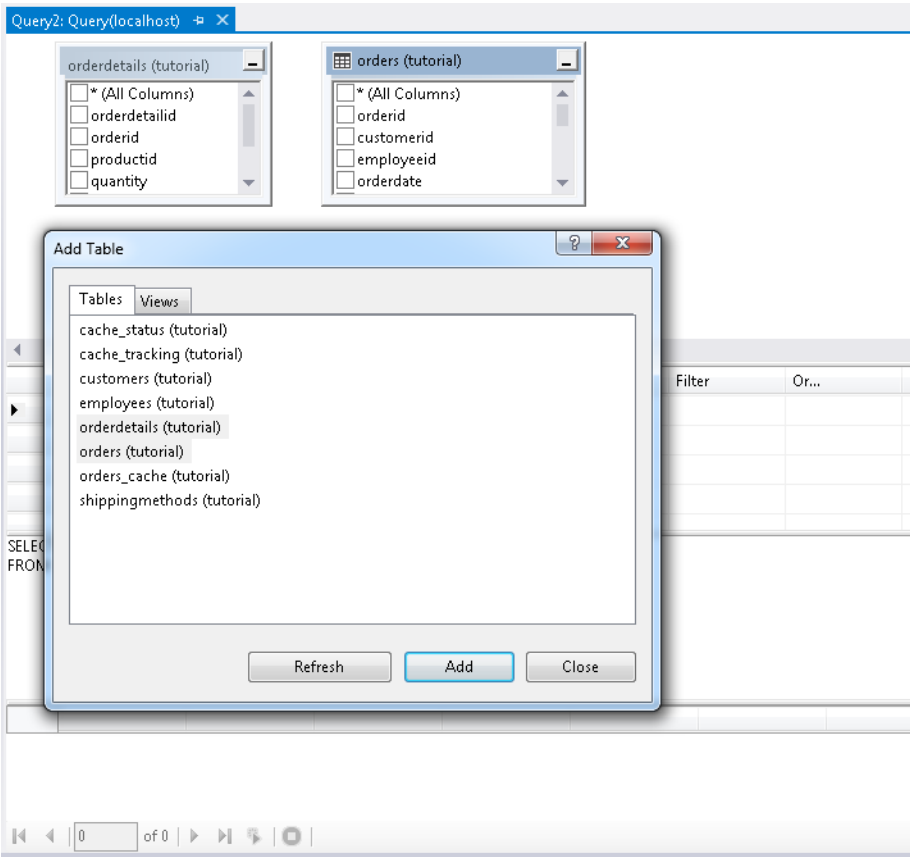
1. Use the Visual Studio Server Explorer context menu for the corresponding node.
2. You can modify any of the settings by overwriting the existing values with new ones.

## 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

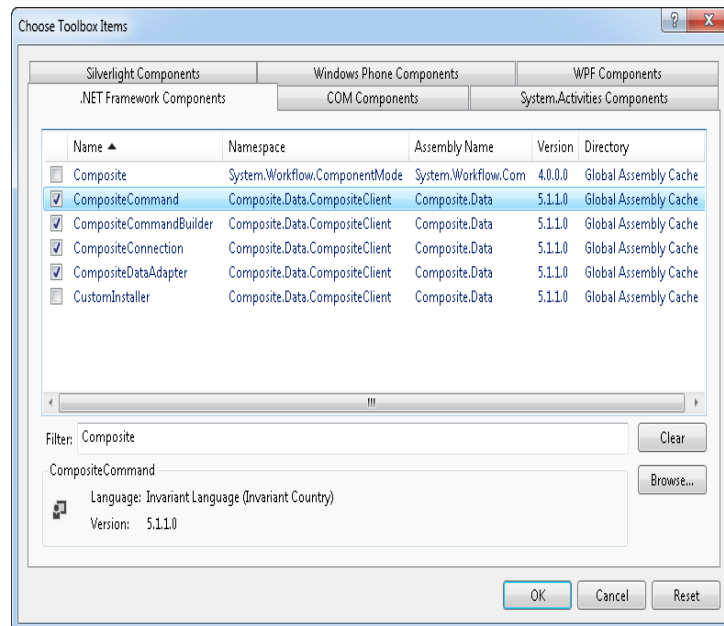
TDV ADO.NET components can be added to the Visual Studio .NET Toolbox. After the objects are in the toolbox, Visual Studio Designer lets you add objects to the Windows Forms so that then you can configure them using either the Properties windows or a wizard. The following components are available:

- CompositeConnection
- CompositeCommand
- CompositeCommandBuilder
- CompositeDataAdapter

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

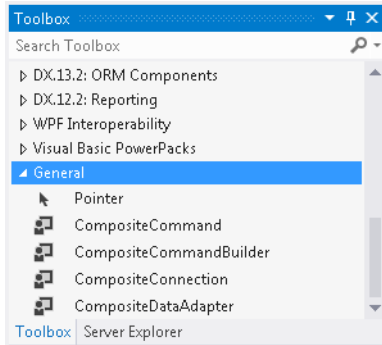
### To use the ToolBox items

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



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.

## 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 145](#)
- [Create a CompositeCommand Object, page 146](#)
- [Select Data from a TDV Published Resource, page 147](#)
- [Getting the Column Type, page 149](#)
- [Getting Column Metadata, page 150](#)
- [Using an Update Operation in the Sample Code, page 151](#)
- [About Using Parameters, page 152](#)
- [Invoking a Stored Procedure Example, page 154](#)
- [Example with Special Data Types, page 158](#)
- [Retrieving Metadata, page 160](#)



- [Retrieving Tables with a Named Schema, page 161](#)

## 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=composite;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 147](#)
- [Using the conn.CreateCommand Method to Create the CompositeCommand Object, page 147](#)
- [Using CompositeCommand to Create the CompositeCommand Object, page 147](#)

## 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 152](#), 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;  
}  
}
```