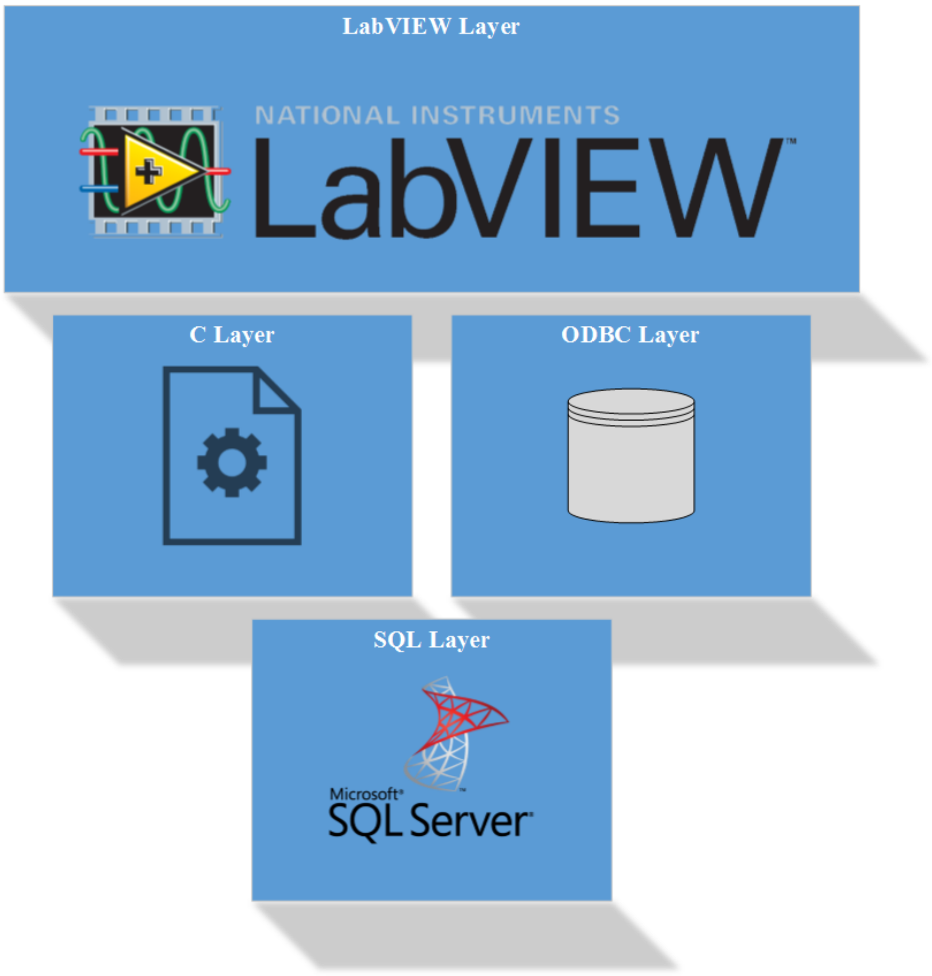
Database Connectivity Toolkit for Big Data

User Manual



Ovak Technologies
2015

Contents

1. Introduction

1.1. Definitions and Acronyms

SQL – Structured Query Language;

DB – Database;

LV – LabVIEW;

DLL – Dynamic Link Library;

BCD – Binary-coded decimal;

DBMS – Database Management System;

ODBC – Open Database Connectivity;

ADO – ActiveX Data Objects;

OLE DB – Object Linking and Embedding database;

CLI – Command-Line Interface;

DSN – Data Source Name;

MDAC – Microsoft Data Access Components.

1.2. Purpose

This manual contains information about how to use *Database Connectivity Toolkit for Big Data* to communicate, create, store, modify, retrieve and manage data in database management system using Structured Query Language.

The manual requires that you have basic understanding of the LabVIEW environment, your computer operating system and SQL.

1.3. Overview

Database Connectivity Toolkit for Big Data is a library, which consists of low-level DLL functions, which you can perform both common database tasks and advanced customized tasks. Generally, the toolkit consists of Low-level DLL written in C programming language and meant to establish fast and reliable communication with DBMS using ODBC 3.x API .

The following list describes the main features of the toolkit:

- Works with any database driver that complies with ODBC.
- Maintains a high level of portability. In many cases, you can port an application to another database by changing the connection information you pass to the *"Connect.vi"*;
- Converts database column values from native data types to standard ODBC 3.x API data types.
- Permits the use of SQL statements with all ODBC supported database systems;
- Provides multiuser access to MS SQL databases.

The main advantages of the toolkit are:

- High-performance connectivity for MS SQL Databases;
- Up to 10x fast transactions;
- Developer friendly data formats;
- Recommended for web-based SCADA systems development.

Due to the wide range of databases with which the *Database Connectivity Toolkit for Big Data* works, some portability issues remain.
Consider the following issues when choosing your database system:

- Some database systems, particularly, the flat-file databases such as dBase, do not support floating-point numbers. In cases where floating-point numbers are not supported, the toolkit converts floating-point numbers to the nearest equivalent, usually binary-coded decimal, before storing them in the database. Very large or very small floating-point numbers can pass the upper or lower limits of the precision available for a BCD value.
- Restrictions on column names vary among database systems. For maximum portability, limit column names to ten uppercase characters without spaces. You might be able to access longer column, or field, names or names that contain spaces by enclosing the name in double quotes.
- Some database systems do not support date, time, or date and time data types.

2. Open Database Connectivity (ODBC)

MDAC includes ODBC, OLE DB, and ADO components. ODBC is designed for maximum *interoperability* - that is, the ability of a single application to access different database management systems with the same source code. *Database Connectivity Toolkit for Big data* call functions in the ODBC interface, which are implemented in database-specific modules called *drivers*. The use of drivers isolates *Database Connectivity Toolkit for Big data* from database-specific calls in the same way that printer drivers isolate word processing programs from printer-specific commands. As drivers are loaded at run time, a user needs only to add a new driver to access a new DBMS; it is not necessary to recompile or relink the application. Primarily, ODBC is a specification for a database API. This API is independent of any other DBMS or operating system. C programming language was used for this toolkit with ODBC API which is language-independent. The ODBC API is based on the CLI specifications from Open Group and ISO/IEC. ODBC 3.*x* fully implements both of these specifications — earlier versions of ODBC were based on preliminary versions of these specifications but did not fully implement them, but in the ODBC 3.*x* there are some features commonly needed by developers of screen-based database applications, such as scrollable cursors. ODBC was created to provide a uniform method of access to different or heterogeneous DBMSs.
The ODBC architecture consists of four components:

- **Application** performs processing and calls ODBC functions to submit SQL statements and retrieve results.
- **Driver Manager** loads and unloads drivers on behalf of an application, processes ODBC function calls or passes them to a driver.
- **Driver** processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated DBMS.

- **Data Source** consists of the data the user wants to access and its associated operating system, DBMS, and network platform (if any) used to access the DBMS.

The hierarchy of data interface layers between LabVIEW and a database using ODBC API is presented in Figure 1.

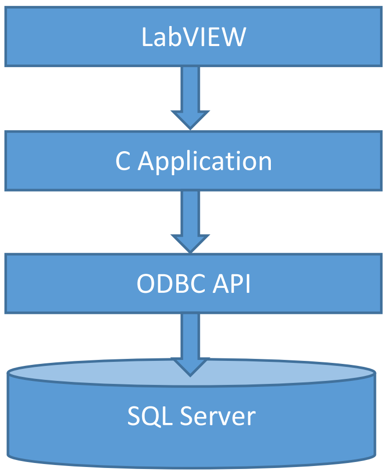


Figure 1 The hierarchy of data interface layers

3. Registering ODBC Driver

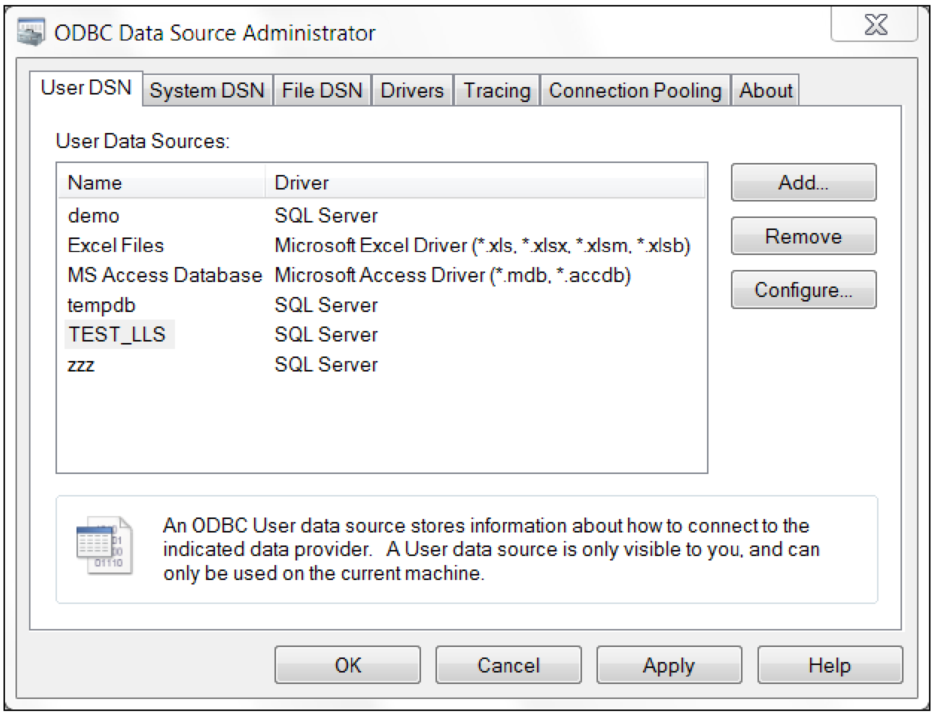
3.1. Opening the ODBC Driver Manager

In order to access the registered ODBC driver entries you need to take the following steps:

Note: This configuration is designed for Windows 7.

1. Click the *«Start»* button on the Windows task bar;
2. Select *«Control Panel»*;
3. Make sure *«Category»* is selected in *«View by»* menu;
4. Selectt *«System and Security»*;
5. Click on *«Administrative Tools»*;
6. Double-click on *«Data Sources (ODBC)»*.

The dialog box shown in figure 2 should be displayed.


ODBC Data Source Administrator

User DSN | System DSN | File DSN | Drivers | Tracing | Connection Pooling | About

User Data Sources:

Name	Driver
demo	SQL Server
Excel Files	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)
MS Access Database	Microsoft Access Driver (*.mdb, *.accdb)
tempdb	SQL Server
TEST_LLS	SQL Server
zzz	SQL Server

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

Figure 2 ODBC Data Source Administrator

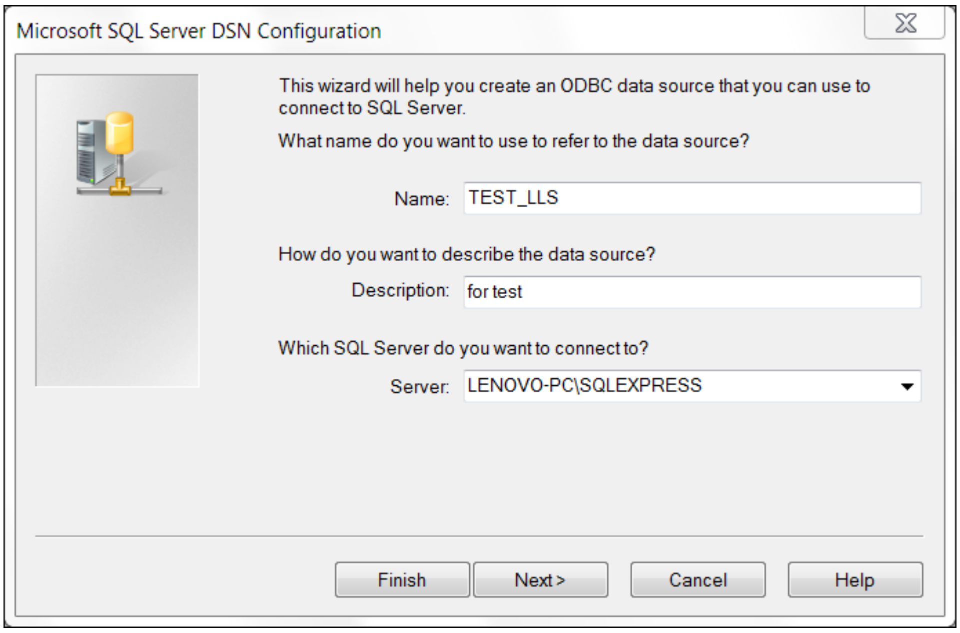
3.2. Configuring the default DSN entry

To configure the default DSN entry for TEST_LLS ODBC you need to make the following steps:

1. Highlight the entry named «TEST_LLS» on one of the DSN tabs («User DSN», «System DSN» or «File DSN»).
2. Click on *«Configure»*.

Note: Each tab has its own default entry for RDM. The changes made in entry of one tab will not affect those in other tabs.

3. The *«Microsoft SQL Server DSN Configuration»* dialog box will be displayed.


Microsoft SQL Server DSN Configuration

This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name: TEST_LLS

How do you want to describe the data source?

Description: for test

Which SQL Server do you want to connect to?

Server: LENOVO-PC\SQLEXPRESS

Figure 3 Microsoft SQL Server DSN Configuration

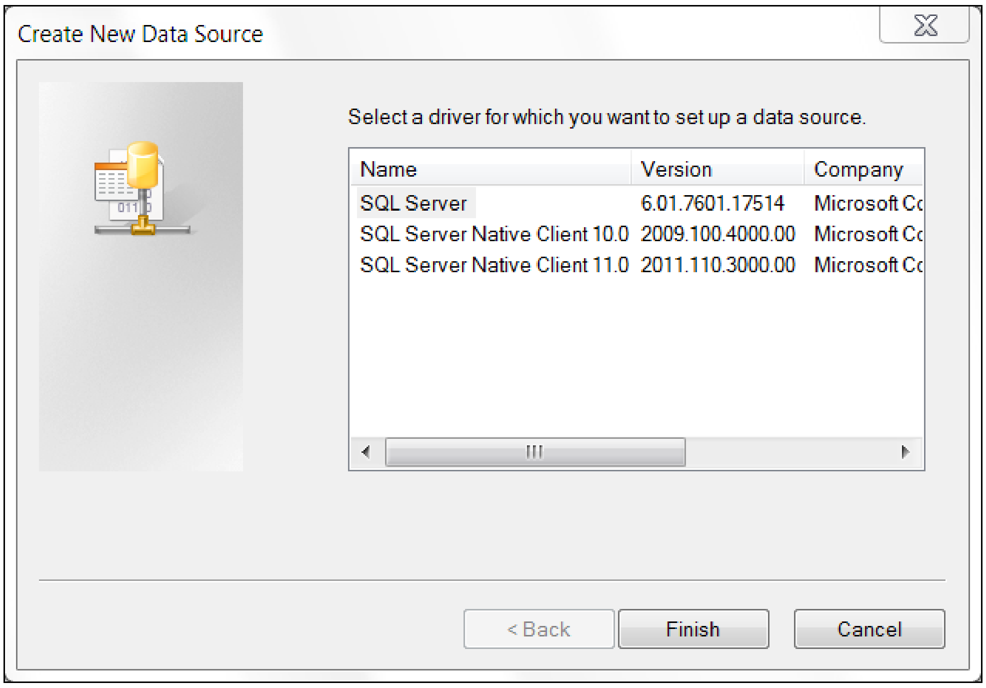
The «*Microsoft SQL Server DSN Configuration*» consists of the following items:

1. **Data source name** – data source name is a unique identifier of the DSN entry. The default value is *«TEST_LLS»*. In order to change the value, type the desired text in the edit box and click on *«Finish»*.
2. **Description** – description is a text of the DSN entry. In order to change the value, type the desired text in the edit box and click on *«Finish»*.
3. **Databases** – ODBC Driver requires that one or more default databases will be specified with ODBC DM. Once it successfully connects to the SQL server, ODBC will open the specified databases. To add, modify or remove default databases, click on *«Databases»*. The *«Databases Configuration»* dialog will be displayed (Figure 3).
4. **Options** – click on the *«Next»* button to configure the advanced options, such as the driver connection type.

3.3. Adding a new DSN entry

To add a new DSN entry for SQL ODBC you need to make the following steps:

1. Highlight the entry named *«TEST_LLS»* on one of the DSN tabs («User DSN», «System DSN" or «File DSN»).
2. Click on *«Add»*.
3. The *«Create New Data Source»* dialog box will be displayed.



Select a driver for which you want to set up a data source.

Name	Version	Company
SQL Server	6.01.7601.17514	Microsoft Cc
SQL Server Native Client 10.0	2009.100.4000.00	Microsoft Cc
SQL Server Native Client 11.0	2011.110.3000.00	Microsoft Cc

Figure 4 Create New Data Source

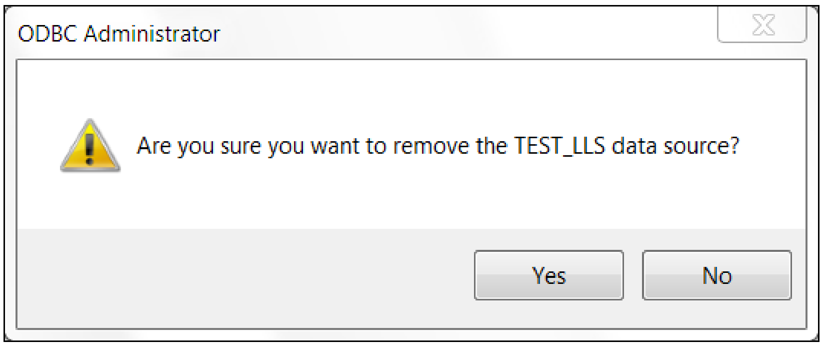
Highlight *«TEST_LLS»* and click *«Finish»*. The *«ODBC SQL Setup»* dialog box (Figure 3) will be displayed.

See the Configuring the default DSN Entry" section for details on the options.

3.4. Removing an existing DSN entry

To remove an existing DSN entry for *«TEST_LLS»* ODBC you need to:

1. Highlight the entry you wish to remove on one of the DSN tabs («User DSN», «System DSN» or «File DSN»)
2. Click on *«Remove»*. The confirmation dialog will be displayed.


ODBC Administrator

Are you sure you want to remove the TEST_LLS data source?

Figure 5 ODBC Administrator

3. Click *«Yes»* to remove the selected DSN entry.

4. Connecting to a Database

Before you can access data in the table or execute SQL statements, you must establish a connection to a database. The *Database Connectivity Toolkit for Big Data* supports DSN string connection to a single database or to multiple databases. Use the *«Connect.vi»* to establish the connection with a database. It is here when all errors are identified because each DBMS uses different parameters for the connection and different levels of security. The different standards also use different methods of connecting to databases. ODBC uses DSN for the connection.

In order to connect to the database via *Database Connectivity Toolkit for Big Data* the developer need to specify the following parameter – DSN String.

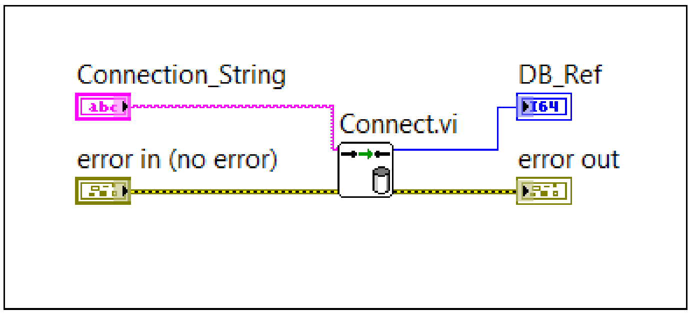


Figure 6 Connect to Database

Some DBMS requires that this parameter should be set in order to connect to the database. You should be familiar with your DBMS and be aware of how to specify the connection parameter.

4.1. DSNs and Data Source Types

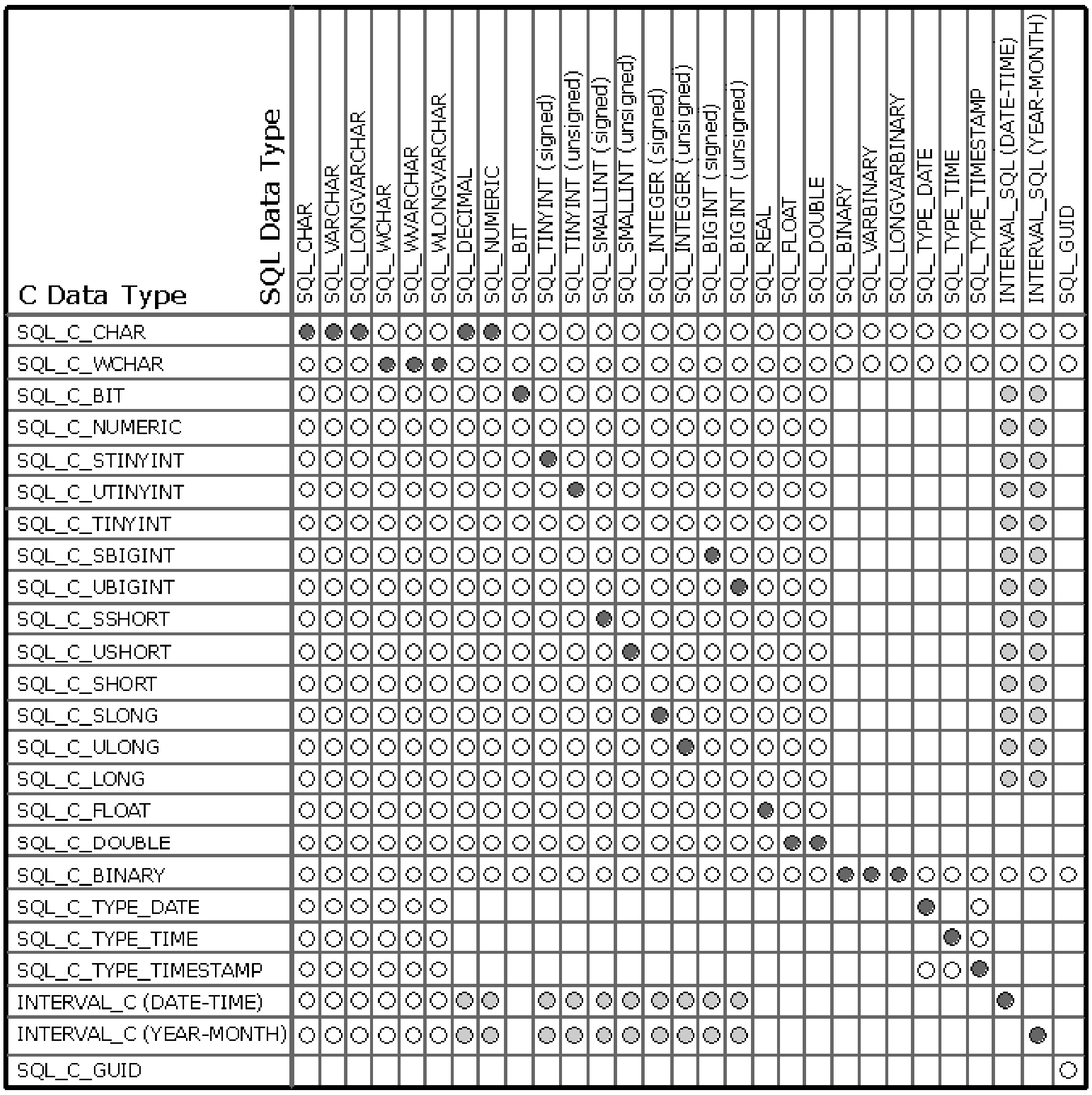
A DSN is the name of the data source, or database, to which you are connecting. The DSN also contains information about the ODBC driver and other connection attributes including paths, security information, and read-only status of the database.

There are two main types of DSNs:

1. **Machine DSNs** are in the system registry and apply to all users of the computer system or to a single user. DSNs that apply to all users of a computer system are system DSNs. DSNs that apply to single users are user DSNs.
2. **File DSNs** is a text file with a *«.dsn»* extension and is accessible to anyone with proper permissions. File DSNs are not restricted to a single user or computer system. Use the ODBC Data Source Administrator to create and configure DSNs.

5. Data Types Mapping

The *Database Connectivity Toolkit for Big Data* maps the various C language data types to data types supported by some of the common DBMS. Figure 7 shows which SQL data types the *Database Connectivity Toolkit for Big Data* C language DLL supports.



C Data Type / SQL Data Type	SQL_CHAR	SQL_VARCHAR	SQL_LONGVARCHAR	SQL_WCHAR	SQL_WVARCHAR	SQL_WLONGVARCHAR	SQL_DECIMAL	SQL_NUMERIC	SQL_BIT	SQL_TINYINT (signed)	SQL_TINYINT (unsigned)	SQL_SMALLINT (signed)	SQL_SMALLINT (unsigned)	SQL_INTEGER (signed)	SQL_INTEGER (unsigned)	SQL_BIGINT (signed)	SQL_BIGINT (unsigned)	SQL_REAL	SQL_FLOAT	SQL_DOUBLE	SQL_BINARY	SQL_VARBINARY	SQL_LONGVARBINARY	SQL_TYPE_DATE	SQL_TYPE_TIME	SQL_TYPE_TIMESTAMP	INTERVAL_SQL (DATE-TIME)	INTERVAL_SQL (YEAR-MONTH)	SQL_GUID
SQL_C_CHAR	●	●	●	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_C_WCHAR	○	○	○	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_C_BIT	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○							◐	◐	
SQL_C_NUMERIC	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○							◐	◐	
SQL_C_STINYINT	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○							◐	◐	
SQL_C_UTINYINT	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○							◐	◐	
SQL_C_TINYINT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○							◐	◐	
SQL_C_SBIGINT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○							◐	◐	
SQL_C_UBIGINT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○							◐	◐	
SQL_C_SSHORT	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○							◐	◐	
SQL_C_USHORT	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○							◐	◐	
SQL_C_SHORT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○							◐	◐	
SQL_C_SLONG	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○							◐	◐	
SQL_C_ULONG	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○							◐	◐	
SQL_C_LONG	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○							◐	◐	
SQL_C_FLOAT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○									
SQL_C_DOUBLE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●									
SQL_C_BINARY	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	●	○	○	○	○	○	○
SQL_C_TYPE_DATE	○	○	○	○	○	○																		●		○			
SQL_C_TYPE_TIME	○	○	○	○	○	○																			●	○			
SQL_C_TYPE_TIMESTAMP	○	○	○	○	○	○																		○	○	●			
INTERVAL_C (DATE-TIME)	○	○	○	○	○	○	◐	◐		◐	◐	◐	◐	◐	◐	◐	◐										●		
INTERVAL_C (YEAR-MONTH)	○	○	○	○	○	○	◐	◐		◐	◐	◐	◐	◐	◐	◐	◐											●	
SQL_C_GUID																													○

● Default Conversion

○ Supported Conversion

◐ Valid only for interval types with just a single field

Figure 7 Data Types

All LabVIEW data types are supported but not necessarily in their native form. For example, bytes (U8 and I8) and words (U16 and I16) can be treated as longs (I32). The binary data type encompasses any piece of LabVIEW data, such as waveform, cluster, or array data that cannot be represented natively in the database. Figure 8 lists LabVIEW data types and the data types in the *Database Connectivity Toolkit for Big Data* to which they correspond.

№	**LabVIEW data types**	**Database Connectivity Toolkit for Big Data data types**
1	8-bit integers	Integer
2	16-bit integers	Integer
3	32-bit integers	Integer
4	8-bit enum	Integer
5	16-bit enum	Integer
6	32-bit enums	Integer
7	64-bit integers	Long
8	64-bit enums	Long
9	Single numeric	Float
10	Double numeric	Double
11	Boolean	Integer
12	String	Char*
13	Date/Time string	Char*
14	Time stamp	Char*
15	Path	Char*
16	I/O channel	Char*
17	Refnum	Integer
18	Complex numeric	Binary
19	Extended numeric	Binary
20	Picture control	Binary
21	Array	Binary
22	Cluster	Struct
23	Variant	Binary
24	Waveform	Binary
25	Digital waveform	Binary
26	Digital data	Binary
27	WDT	Binary
28	Fixed-point numeric	Binary

Database Connectivity Toolkit for Big Data supports refnums, which are ephemeral constructs whose values are meaningless after usage. If you want to save a refnum in the database table, you must first type cast the refnum to an integer and then write the integer in the table.

6. Working with Date/Time Data Types

Date/time is an important data type for databases. You can use the time stamp data type to represent date and time in LabVIEW. *Database Connectivity Toolkit for Big Data* can convert the LabVIEW date time type to MS SQL date time type. The main problem with the date/time data type is that there is no uniformity and each database supports a different format. In other words, when you select date/time values from a database, they might be returned in a different form depending on the DBMS.

7. Handling NULL Values

Databases have *NULL* fields that are empty fields containing no data. *Database Connectivity Toolkit for Big Data* treats *NULLs* as default data, such as an empty string, a zero-value numeric, or a *FALSE* Boolean. Therefore, for example, you cannot easily differentiate between a 0.00 value in a numeric from one that is *NULL*. When you convert the *NULL* values in the variant array into numeric values, the *NULLs* become 0 values. However, when you convert the variant array into strings, the *NULLs* become empty strings.

8. Performing Standard Database Operations

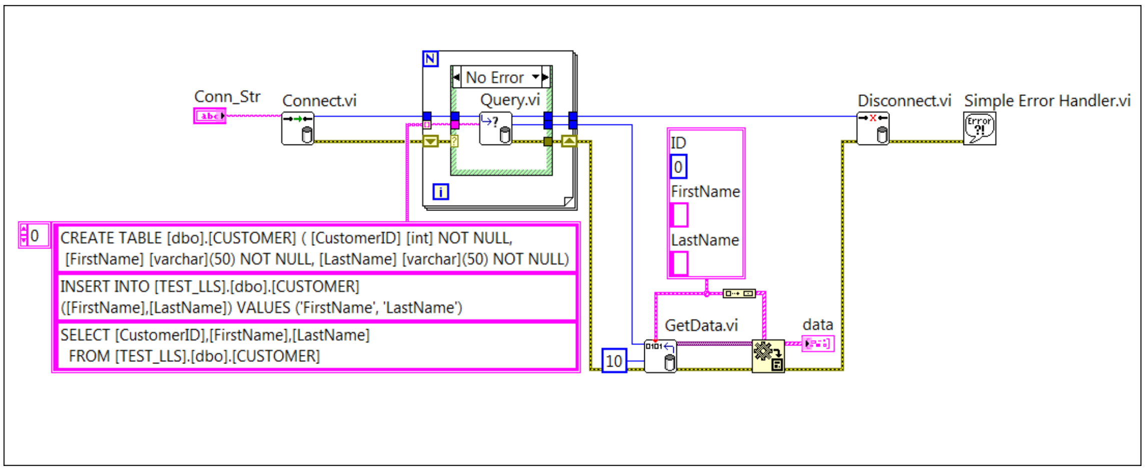
You can use the *Database Connectivity Toolkit for Big Data* VIs to write data to or read data from databases, create and delete tables.

Creating or deleting table, writing or reading data are similar with the *Database Connectivity Toolkit for Big Data*. You open a connection, execute queries, get database data, and close the connection when you are finished. Figure 8 shows the block diagram of a VI that create table, write test information to a database table, read data from a database table and then converts the data to the appropriate data types in LabVIEW. The connection information is a ODBC connection string (DSN).

Four databases VIs are represented in Figure 8:

1. Toolkit's Connect VI
2. Toolkit's Query VI
3. Toolkit's GetData VI
4. Toolkit's Disconnect VI.

LabVIEW data types are converted to the appropriate database data types.



```
CREATE TABLE [dbo].[CUSTOMER] ( [CustomerID] [int] NOT NULL,
 [FirstName] [varchar](50) NOT NULL, [LastName] [varchar](50) NOT NULL)
INSERT INTO [TEST_LLS].[dbo].[CUSTOMER]
([FirstName],[LastName]) VALUES ('FirstName', 'LastName')
SELECT [CustomerID],[FirstName],[LastName]
  FROM [TEST_LLS].[dbo].[CUSTOMER]
```

Figure 8 Standard Database Operations

Note: The database data is returned as a one-dimension array of variants (Figure 8).

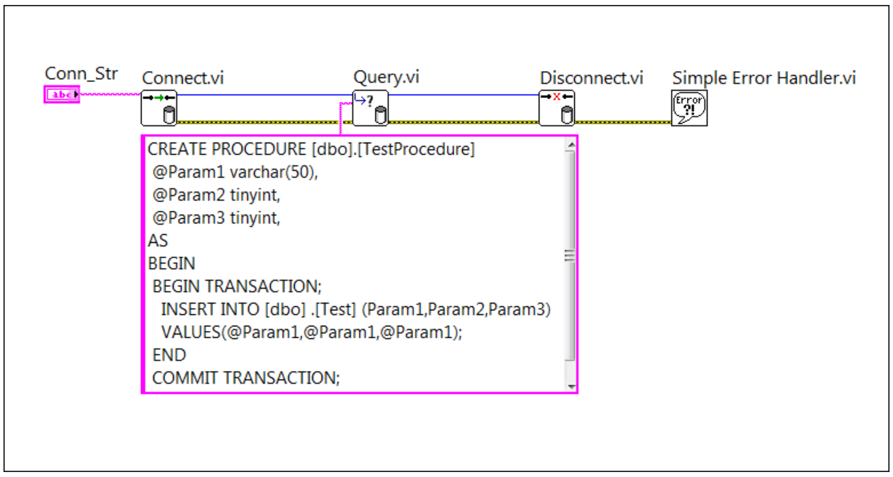
9. Using Stored Procedures

A stored procedure is a precompiled collection of SQL statements and optional control-of-flow statements, similar to a macro. Each database and data provider supports stored procedures differently. For example, you can create a stored procedure using the Jet 4.0 provider, but Access does not support stored procedures through its usual user interface. A stored procedure created in one DBMS might not work with another. You can use the *Database Connectivity Toolkit for Big Data* to create and run stored procedures, both with and without parameters.

Although using stored procedures is an advanced task, stored procedures offer the following benefits to your database applications:

- **Performance** – stored procedures are usually more efficient and faster than regular SQL queries because SQL statements are analyzed for syntactical accuracy and precompiled by the DBMS when the stored procedure is created. In addition, combining a large number of SQL statements with conditional logic and parameters into a stored procedure allows the procedures to perform queries, make decisions, and return results without extra trips to the database server.
- **Maintainability** – stored procedures isolate the lower-level database structure from the LabVIEW application. As long as the table names, column names, parameter names, and types do not change from what is stated in the stored procedure, you do not need to modify the procedure when changes are made to the database schema. Stored procedures are also a way to support modular SQL programming because after you create a procedure, you and other users can reuse that procedure without knowing the details of the tables involved.
- **Security** – when creating tables in a database, the Database Administrator can set EXECUTE permissions on stored procedures without granting SELECT, INSERT, UPDATE, and DELETE permissions to users. Therefore, the data in these tables is protected from users who are not using the stored procedures.

You usually create stored procedures in the DBMS environment. Some DBMSs, such as SQL Server, contain a library of system-stored procedures that perform common administrative tasks with databases. For Creating «Stored Procedure» you can use *Database Connectivity Toolkit for Big Data*, as shown in Figure 9.



```
CREATE PROCEDURE [dbo].[TestProcedure]
 @Param1 varchar(50),
 @Param2 tinyint,
 @Param3 tinyint,
AS
BEGIN
 BEGIN TRANSACTION;
  INSERT INTO [dbo] .[Test] (Param1,Param2,Param3)
  VALUES(@Param1,@Param1,@Param1);
 END
 COMMIT TRANSACTION;
```

Figure 9 Creating Stored Procedure

For creating «Stored Procedure» the same Vis are used as you to perform a typical SQL query. However, the syntax of the SQL query string is different. The SQL query string is a stored procedure.

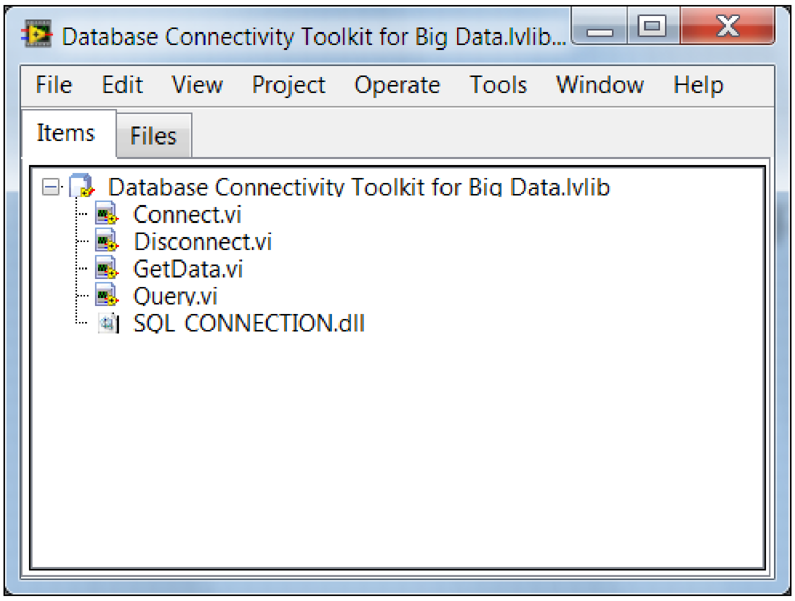
10.Running Stored Procedures

You can run a stored procedure by inserting the name of the procedure as an SQL query. Stored procedures can use variables internally as well as pass parameters into and out of the procedure. You can use parameters with stored procedures in two ways. In the first method, you build SQL query strings that contain the name of the stored procedure with the values embedded at the appropriate places in the query.

11.Architecture

The data type Converts are defined in *GetData VI*, which is included in *Database Connectivity Toolkit for Big Data* library. The SQL Connection is defined in DLL, which is included in *Database Connectivity Toolkit for Big Data Library.*
Database Connectivity Toolkit for Big Data Library has the following functions:

- Connect.vi – opens connection to SQL use ODBC Connection String (DSN);
- Disconnect.vi – closes connection to SQL;
- GetData.vi – this VI gets data from Database using data reference;
- Query.vi – this VI executes query in SQL database.
- SQL_CONNECTION.dll – this is a *DLL*, which is written in C programing language. It contains *Connect*, *Query*, *GetData* and *Close* functions.



Database Connectivity Toolkit for Big Data.lvlib
Connect.vi
Disconnect.vi
GetData.vi
Query.vi
SQL CONNECTION.dll

Figure 10 Database Connectivity Toolkit for Big Data Library

12.Example

Complete the following steps to run this example:

1. In the **Project Explorer** window, open **Example.vi.**
2. Enter **Connection String** (DSN) parameter, the query and data types cluster.
3. Click **RUN** button.

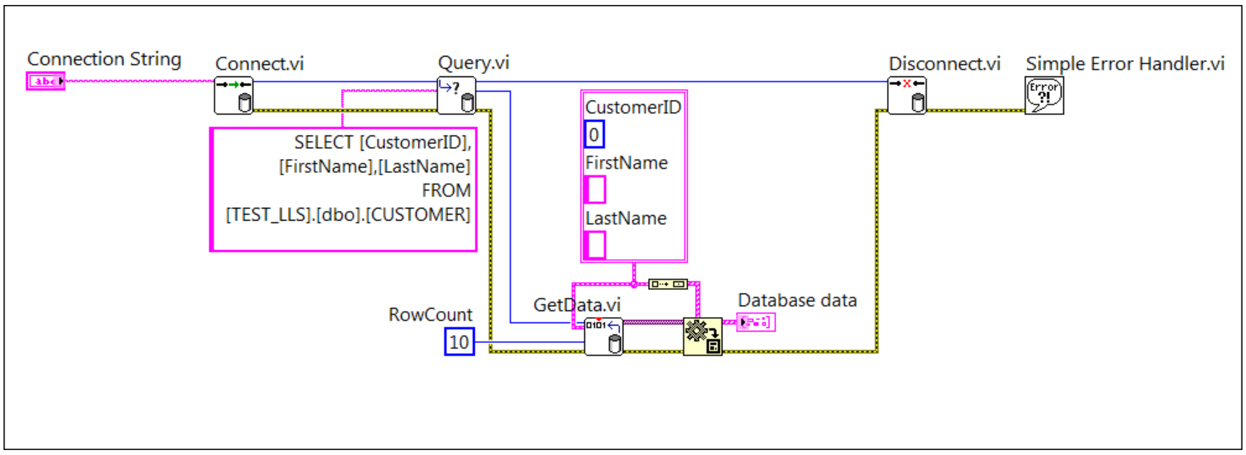
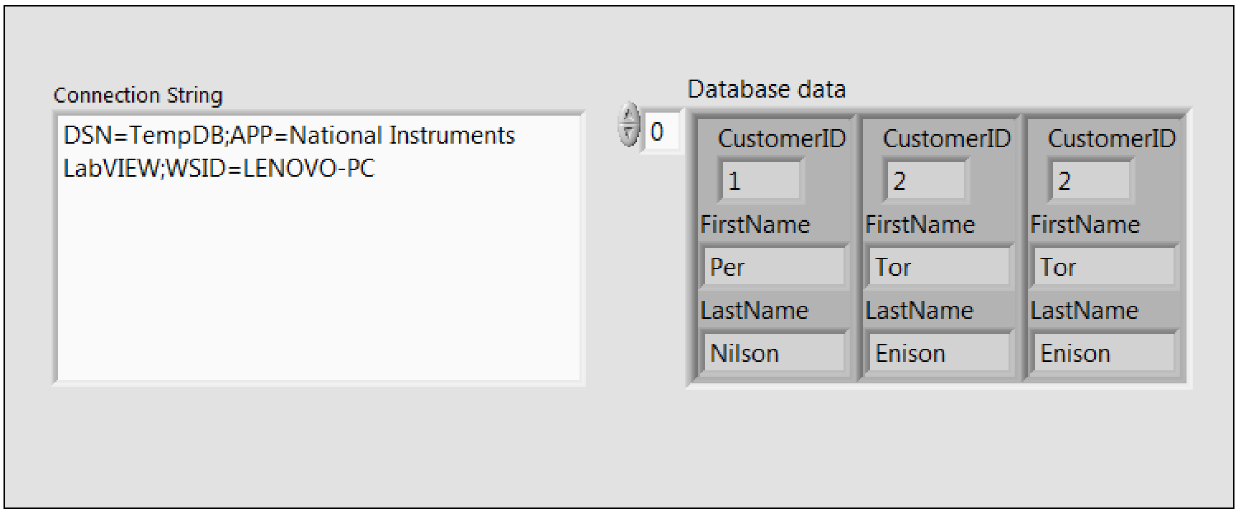


Figure 11 Example Block Diagram.



Connection String

DSN=TempDB;APP=National Instruments LabVIEW;WSID=LENOVO-PC

Database data

0

CustomerID	CustomerID	CustomerID
1	2	2
FirstName	FirstName	FirstName
Per	Tor	Tor
LastName	LastName	LastName
Nilson	Enison	Enison

Figure 12 Example Front Panel

13.Error Codes List

№	Error Code	Description
1	SQL_SUCCESS	Function is completed successfully. The application calls *SQLGetDiagField* to retrieve additional information from the header record (0).
2	SQL_SUCCESS_WITH_INFO	Function is completed successfully, possibly with a nonfatal error (warning). The application calls *SQLGetDiagRec* or *SQLGetDiagField* to retrieve additional information (1).
3	SQL_ERROR	Function is failed. The application calls SQLGetDiagRec or SQLGetDiagField to retrieve

		additional information. The contents of any output arguments to the function are undefined (2).
4	SQL_INVALID_HANDLE	Function is failed due to an invalid environment, connection, statement, or descriptor handle. This indicates a programming error. No additional information is available from *SQLGetDiagRec* or *SQLGetDiagField*. This code is returned only when the handle is a null pointer or is the wrong type, such as when a statement handle is passed for an argument that requires a connection handle (3).
5	SQL_NO_DATA	No more data was available. The application calls *SQLGetDiagRec* or *SQLGetDiagField* to retrieve additional information (4).
6	SQL_NEED_DATA	More data is needed, such as when parameter data is sent at execution time or additional connection information is required. The application calls *SQLGetDiagRec* or *SQLGetDiagField* to retrieve additional information, if any exists (5).

14.System Requirements

LabVIEW Base, Full, or Professional Development System

15.LabVIEW Features and Concepts Used

- Case structures
- Clusters
- Enums
- Error clusters
- Shift registers
- While Loops
- For Loops
- States

16.Support Information

For technical support, please, contact Ovak Technologies at:
Phone: + 374 (010) 21-97-68

Email: support@ovaktechnologies.com

Web: www.ovaktechnologies.com