insightsoftware

Simba SDK

Build a C++ ODBC Driver in 5 Days (Windows)

Version 10.3

August 2024

Copyright

This document was released in August 2024.

Contact Us

insightsoftware

www.insightsoftware.com

Table of Contents

Introduction

The Build a C++ ODBC Driver in 5 Days guide demonstrates how to create a custom ODBC driver using SimbaEngine. Procedures in the guide explain how to modify and customize the Quickstart sample driver included with SimbaEngine.

At the end of five days, you will have a read-only driver that connects to your data store.

About SimbaEngine

ODBC is one of the most established and widely supported APIs for connecting to and working with databases. The ODBC specification provides a standard interface to which any ODBC-enabled application can connect. At the heart of ODBC technology is the driver, which connects an application to a data store.

SimbaEngine is a complete implementation of the ODBC specification. The libraries of SimbaEngine hide the complexity of error checking, session management, data conversions and other low-level implementation details, exposing a simple API called the Data Store Interface API (DSI API) that defines the operations needed to access a data store.

You use SimbaEngine to create an executable file that common reporting applications access when connecting to your data store in the process of executing an SQL statement. The executable file can be a Windows DLL; a Linux, Unix or Mac OS X shared object; a stand-alone server; or, some other form of executable. First, you create a DSI implementation (DSII) customized as needed to connect to your data source. Then, you create the executable by linking libraries from SimbaEngine with your custom DSI implementation. The project files or make files link in the appropriate SimbaODBC and SimbaEngine libraries to complete the driver. In the final executable, the components from SimbaEngine take responsibility for meeting the data access standards while your custom DSI implementation takes responsibility for accessing your data store and translating it to the DSI API.

Note: For a brief description of the ODBC standard, see http://www.simba.com/resources/data-access-standards-library#!odbc. For complete information on the ODBC 3.80 specification, see the ODBC Programmer's Reference at http://msdn.microsoft.com/en-us/library/ms714177(v=vs.85).aspx. For full documentation on SimbaEngine, see http://www.simba.com/products/simba-engine-sdk.

About the Quickstart Sample Driver

SimbaEngine includes sample drivers that you can use as a starting point for customizing your own driver. The Quickstart driver is a sample DSI implementation of an ODBC driver written in C++ that reads files in tabbed Unicode text format. The Simba SQLEngine component is required to perform the necessary SQL processing in the implementation because text files are not a SQL-aware data source.

Using the Quickstart sample driver to prototype a DSI implementation for your own data store helps you learn how SimbaEngine works. If you remove the shortcuts and simplifications implemented in the Quickstart driver, you can also use the driver as the

foundation for a commercial DSI implementation. Using the Quickstart sample driver is a fast and effective way to prepare a data access solution for your customers.

The UML diagram in About the Quickstart Sample Driver shows a typical design pattern for a DSI implementation.

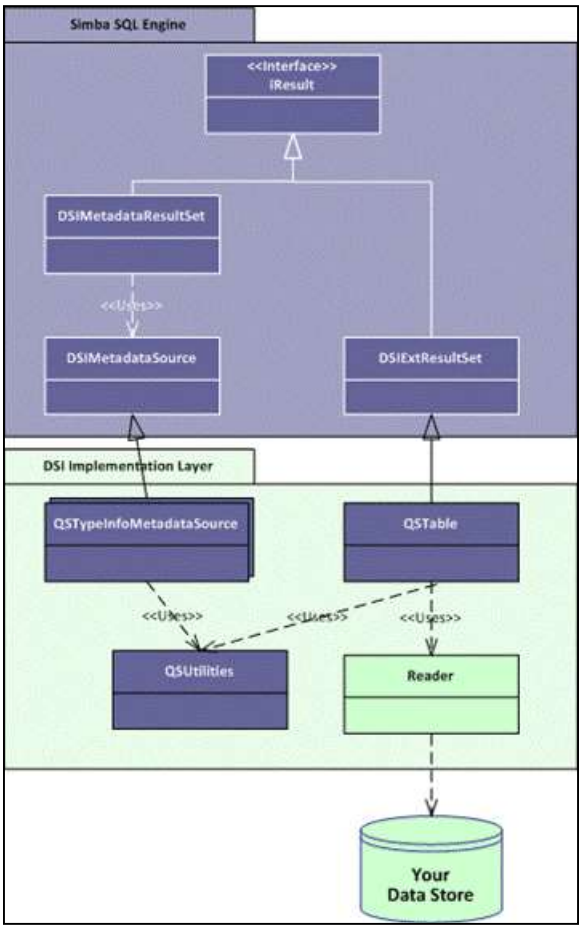

Figure 1: Design Pattern for a DSI Implementation

Notice the circular pattern of class relationships, headed by iResult and anchored by QSUtilities. The iResult class is responsible for retrieving column data and maintaining a cursor across result rows. The QSUtilities class contains platform-specific utility functions.

To implement data retrieval, your Reader class interacts directly with your data store to retrieve data, and then deliver the data to the QSTable class on demand. The Reader class should manage caching, buffering, paging and all the other techniques to speed data access.

As a starting point, to make your driver work properly with Microsoft Excel you can add metadata access by implementing the QSTypeInfoMetadataSource class and using the DSIExtMetadataHelper class. The DSIExtMetadataHelper class is responsible for iterating through tables and stored procedures so the engine can generate catalog function metadata.

Overview

To implement a working DSI with your data store, you need to:

1. Set up the development environment
2. Make a connection to the data store
3. Retrieve metadata
4. Work with columns
5. Retrieve data

In the Quickstart sample driver, you can identify areas of the code requiring modification by referring to pragma messages containing sequentially numbered TODO labels and a brief description.

Some areas of code you need to modify relate to retrieving data and metadata from your data store into the Simba SQLEngine. The Quickstart sample driver includes the classes and code to access the example data store. You need to modify the sample code to access your data store.

Some modifications you may make to Quickstart sample driver code relate to productization, including:

- Naming the driver
- Setting configuration properties for the driver
- Naming the XML error file and log files

Day One – Windows Instructions

To set up and prepare your development environment to create a driver based on the Quickstart sample driver:

1. Install **SimbaEngine**.
2. Build the Quickstart sample driver.
3. Review the keys that the Quickstart sample driver adds to the **Windows Registry**.
4. In the **ODBC Data Source Administrator**, review the DSNs that the Quickstart sample driver creates.
5. Using the **Quickstart** sample driver, read the tabbed Unicode text format sample data.
6. Create a customizable driver project based on the **Quickstart** sample driver.
7. Test your customizable driver project by reading the tabbed Unicode text format Quickstart sample data.

Detailed instructions for each step appear below.

Installing SimbaEngine

To install SimbaEngine:

1. Uninstall any previous versions of SimbaEngine installed on your computer.
2. Close Visual Studio.
3. Double-click to run SimbaEngine setup executable that corresponds to your version of Visual Studio, and then follow the instructions provided in the installation wizard.

> **Important:** The SimbaEngine environment variables are defined only for the user that ran the installation. If you install SimbaEngine as a regular user and then run Visual Studio as an administrator, SimbaEngine will not work properly.

In the Build a C++ ODBC Driver in 5 Days guide, the directory where you install SimbaEngine is referred to using the placeholder `InstallDir`. The default installation directory is `C:\Simba Technologies`. When working through procedures, ensure that you replace the `InstallDir` placeholder with the appropriate file path.

Building the Quickstart Sample Driver

To build the Quickstart sample driver:

1. In Microsoft Visual Studio 2022, click **File** > **Open**> **Project** or **Solution**.
   Note: Visual Studio 2022 is used as an example, but also supported.
2. In the **Open Project** menu, navigate to the folder `InstallDir\SimbaEngineSDK\\Examples\Source\Quickstart\Source`, then select the file named *`QuickstartDSII_.vcxproj`*, and then click **Open**.

3. Click **Build** > **Configuration Manager**. Click the drop-down arrow next to the **Active Solution Configuration** field, then select **Debug_MTDLL**, and then click **Close**.
4. Click **Build** > **Build Solution**.

OR

Press **F7**

For 32-bit drivers, the build appears in the folder `InstallDir\SimbaEngineSDK\\Examples\ Source\Quickstart\Bin\Win32\Debug_MTDLL`.

For 64-bit drivers, the build appears in the folder `InstallDir\SimbaEngineSDK\\Examples\ Source\Quickstart\Bin\x64\Debug_MTDLL`.

Examining Windows Registry Keys

Installing SimbaEngine adds or updates keys in the Windows Registry to define the sample drivers included with SimbaEngine, as well as Data Source Names (DSNs) using the sample drivers.

For details on information in the Windows Registry related to ODBC drivers, see *Appendex B: Bitness and the Windows Registry* on page 1.

To view the Registry keys related to the Quickstart sample driver:

1. Click **Start**, then type **regedit.exe** in the **Search Programs** and **Files** field, and then press **ENTER**.
2. In the **Registry Editor**, to view the key defining the 32-bit driver on 32-bit Windows or the 64-bit driver on 64-bit Windows, browse to ***HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI/QuickstartDSIIDriver*** and ***HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI/ODBC Drivers*** OR
3. To view the key defining the 32-bit driver on 64-bit Windows, browse to ***HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBCINST.INI/ QuickstartDSIIDriver*** and ***HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ ODBC/ODBCINST.INI/ODBC Drivers***.
4. To view the key defining the DSN using the 32-bit driver on 32-bit Windows or the 64-bit driver on 64-bit Windows, browse to ***HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBC.INI/QuickstartDSII*** OR
5. To view the key defining the DSN using the 32-bit driver on 64-bit Windows, browse to ***HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBC.INI/QuickstartDSII***.
6. Observe the following values set in the **QuickstartDSIIDriver** subkey for the 32-bit driver:
   - **Driver**: `InstallDir\Examples\Builds\Bin\Win32\Release_MTDLL\QuickstartDSII_MTDLL.dll`.
   - **Setup**: `InstallDir\Examples\Builds\Bin\Win32\Release_MTDLL\`

`QuickstartDSII_MTDLL.dll`.

- **Description**: `Sample 32-bit SimbaEngine Quickstart DSII`.

Observe the following values set in the **QuickstartDSIIDriver** subkey for the 64-bit driver:

- **Driver**: `InstallDir\Examples\Builds\Bin\x64\Release_MTDLL\QuickstartDSII_MTDLL.dll`.
- **Setup**: `InstallDir\Examples\Builds\Bin\x64\Release_MTDLL\QuickstartDSII_MTDLL.dll`.
- **Description**: `Sample 64-bit SimbaEngine Quickstart DSII`.

Also in the **ODBCINST.INI** subkey, a string value named **QuickstartDSIIDriver** and having the value Installed appears in the ODBC Drivers subkey.

Observe the following values set in the **QuickstartDSII** subkey for the 32-bit driver:

- **Driver**: `QuickStartDSIIDriver`.
- **DBF**: `InstallDir\Examples\Databases\Quickstart`.
- **Description**: `Sample 32-bit SimbaEngine Quickstart DSII`.

Observe the following values set in the **QuickstartDSII** subkey for the 64-bit driver:

- **Driver**: `QuickStartDSIIDriver`.
- **DBF**: `InstallDir\Examples\Databases\Quickstart`.
- **Description**: `Sample 64-bit SimbaEngine Quickstart DSII`.

Also in the **ODBC.INI** subkey, a string value named QuickstartDSII having the value QuickstartDSIIDriver appears in the ODBC Data Sources subkey.

Viewing DSNs Using ODBC Data Source Administrator

Familiarize yourself with the DSNs related to the Quickstart sample driver using the ODBC Data Source Administrator.

To view Quickstart DSNs in the ODBC Data Source Administrator:

1. Run the Windows ODBC Data Source Administrator.

   For 32-bit drivers on 32-bit Windows and 64-bit drivers on 64-bit Windows, open the Control Panel, select Administrative Tools, and then select Data Sources (ODBC). If your Control Panel is set to view by category, then Administrative Tools is located under System and Security.

   For 32-bit drivers on 64-bit Windows, you must use the 32-bit ODBC Data Source Administrator. You cannot access the 32-bit ODBC Data Source Administrator from the

start menu or control panel on 64-bit Windows. Only the 64-bit ODBC Data Source Administrator is accessible from the start menu or control panel. On 64-bit Windows, to launch the 32-bit ODBC Data Source Administrator you must run C:\WINDOWS\SysWOW64\odbcad32.exe. See *ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit* on page 1 for details.

2. In the **ODBC Data Source Administrator**, click **System DSN tab**.
3. Scroll through the list of System Data Sources, select **QuickstartDSII** and then click **Configure**.
   The **Data Source Configuration** window opens and displays the data source name, description and the data directory.
4. Click **Cancel** to close the **Data Source Configuration** window.

Connecting to the Data Source

To test connecting to the data source using the Quickstart sample driver, you can use any ODBC application, such as Microsoft Excel, Microsoft Access or ODBCTest. In this section, we will use the ODBCTest tool, which is included in the Microsoft Data Access (MDAC) 2.8 Software Development Kit (SDK) available for download from http://www.microsoft.com/downloads/details.aspx?FamilyID=5067faf8-0db4-429a-b502-de4329c8c850&displaylang=en

To test connecting to the data source using the Quickstart sample driver:

1. Start the ODBC Test tool. By default, the ODBC Test application is installed in the following folder: `C:\Program Files (x86)\Microsoft Data Access SDK 2.8\Tools\`

Navigate to the folder that corresponds to your machine's architecture (amd64, ia64 or x86) and then click `odbcte32.exe` to launch the ANSI version or click `odbct32w.exe` to launch the Unicode version.

It is important to run the correct version of the ODBC Test tool for ANSI or Unicode and 32-bit or 64-bit.

2. In the ODBC Test tool, click **Conn**, and then click **Full Connect.** The **Full Connect** window opens.
3. In the Full Connect dialog, select **QuickstartDSII** data source from the list of data sources, and then click **OK**.

If you do not see your data source in the list, ensure that you are running the version of the ODBCTest tool that corresponds to the version of the data source that you created. For example, if you created a 32-bit data source then you need to use the 32-bit version of the ODBCTest tool.

A new window opens displaying a message that you have successfully connected to the data source.

Setting Up a Customizable Project

Now that you have built the example driver, you are ready to set up a development project to create a custom ODBC driver.

> **Important:** Customizing the Quickstart sample driver project directly is not recommended. At times, referring to the original Quickstart sample driver project may help you debug your custom driver. Also, if you customize the Quickstart sample driver project, then your changes may be lost when you upgrade to a new release of SimbaEngine in the future.

To set up a customizable project based on the Quickstart sample driver:

1. In your Windows Explorer window, copy the `[INSTALL_DIRECTORY]\SimbaEngineSDK\ \Examples\Source\Quickstart` directory and paste it to the same location. This will create a new directory called `"Quickstart - Copy"`. Rename the directory to something that is meaningful to you. This will be the top-level directory for your new project and DSI implementation files. For the rest of this tutorial, when you see `<YourProjectName>` in the instructions, replace this with the name you choose for this directory which is also the name of your project.
2. Open your new directory then open the `Source` directory and rename the `QuickstartDSII_.vcproj` file in it to `<YourProjectName>.vcproj` file where you replace `<YourProjectName>` with the name of your project. This will be the project file for your new ODBC driver.
3. Rename the `.sln` file. This new `<YourProjectName>.sln` file is the solution file for your new ODBC driver.
4. Using a text editor, open the project file `(.vcproj)` and replace every instance of `"QuickstartDSII"` in the source code with the name of your new ODBC driver. Then save and close the file.
5. Using a text editor, open the solution file `(.sln)` and replace every instance of "`QuickstartDSII`" in the source code with the name of your new ODBC driver. In addition, the name of the project file must be updated to match the `<YourProjectName>.vcproj` project file that you renamed. Then, save and close the file.

Building Your Customizable Project

To build your customizable ODBC driver project based on the Quickstart sample driver project:

1. Launch Microsoft Visual Studio .
2. Click **File** > **Open** > **Project** or **Solution**.
3. Navigate to `[INSTALL_ DIRECTORY]\SimbaEngineSDK\\Examples\Source\<YourProjectName>\ Source` and then open the `<YourProjectName>.vcproj` file.
4. Click **Build** > **Configuration Manager** and make sure that the active solution configuration is "`Debug_MTDLL`" and then click **Close**.

5. Click **Build** > **Build Solution** or press F7 to build the driver.
   This will build the `Debug_MTDLL` version of the driver and place it in the location: *`[INSTALL_DIRECTORY]\SimbaEngineSDK\\Examples\Source\<YourProjectName>\Bin\Win32\Debug_MTDLL`*.

6. When you build your new project, "TODO" messages appear in the Output window along with the build information.
   If the Output window is not displayed automatically, you can open it by selecting **Debug** > **Windows** > **Output**.

TODO #1: Construct driver singleton.
TODO #2: Set the driver properties.
TODO #3: Set the driver-wide logging details.
TODO #4: Set the connection-wide logging details.
TODO #5: Check Connection Settings.
TODO #6: Establish A Connection.
TODO #7: Create and return your Metadata Sources.
TODO #8: Open A Table.
TODO #9: Register Messages xml file for handling by DSIMessageSource.
TODO #10: Set the vendor name, which will be prepended to error messages.

Over the next four days, you will be visiting each "TODO" and modifying the source code.

Updating the Windows Registry

When you set up a project to create a custom driver based on a sample driver provided with SimbaEngine, you need to create keys in the Windows Registry similar to the keys that the sample driver uses to define the driver as well as DSNs for connecting the driver to your data source. To create keys, SimbaEngine includes .REG files that you can customize and import into the Registry.

To create Registry keys for your custom driver project based on the Quickstart sample driver:

1. In Microsoft Visual Studio , click **File** > **Open** > **File** and navigate to `[INSTALL_DIRECTORY]\SimbaEngineSDK\\Examples\Source\<YourProjectName>\Source`.

2. For 32-bit Windows, open `SetupMyQuickstartDSII-32on32.reg`.

   For a 32-bit ODBC driver on 64-bit Windows, open `SetupMyQuickstartDSII-32on64.reg`.

For a 64-bit ODBC driver on 64-bit Windows, open `SetupMyQuickstartDSII-64on64.reg`.

3. In the file, replace `[INSTALL_DIRECTORY]` with the path to the installation directory. In the path, you must enter double backslashes. For example, by default, the samples are installed to `C:\Simba Technologies` so in that case, replace all instances of `[INSTALL_DIRECTORY]` with `C:\\Simba Technologies`.
4. Update the ODBC Data Sources section to add your new data source. Under the `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources]` section, change `"MyQuickstartDSII=MyQuickstartDSIIDriver` to the name of your new data source and new driver. For example, `"<YourProjectName>DSII"="<YourProjectName>DSIIDriver"`.
5. Modify the data source definition for that data source. Change the line that says `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\MyQuickstartDSII]` so that it contains your new data source name. For example, `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\<YourProjectName>DSII]`.
6. Beside the line that starts with `"Driver"`= enter the path to the driver dll file.
7. Update the ODBC Drivers section to add your new driver. Under the `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers]` section, change `"MyQuickstartDSIIDriver"="Installed"` to match the name of your new driver. For example, `"<YourProjectName>DSIIDriver"="Installed"`.
8. Modify the driver definition for that driver. Change the line that says `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\MyQuickstartDSIIDriver]` so that it contains your new driver name. For example, `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\<YourProjectName>DSIIDriver]`.
9. Besides the line that starts with "`Setup`" and the line that starts with "`Driver`", update the path to the `dll` file.
10. Click **Edit** > **Find and Replace** > **Quick Replace**. Replace "`Quickstart`" in the whole file with the name of your new ODBC driver.
11. Click **Save** and close the file.
12. In the Registry Editor (`regedit.exe`), click **File** > **Import**, navigate to the registry file that you just modified and click **Open**.
    A message is displayed that says that the keys and values have been successfully added to the registry.

Viewing Your DSNs Using ODBC Data Source Administrator

To view the DSNs you created in the Windows Registry for your custom driver project:

1. Run the Windows ODBC Data Source Administrator.

   For 32-bit drivers on 32-bit Windows and 64-bit drivers on 64-bit Windows, open the Control Panel, select **Administrative Tools** > **Data Sources** (ODBC). If your Control Panel is set to view by category, then Administrative Tools is located under **System and Security**.

For 32-bit drivers on 64-bit Windows, you must use the 32-bit ODBC Data Source Administrator. You cannot access the 32-bit ODBC Data Source Administrator from the start menu or control panel on 64-bit Windows. Only the 64-bit ODBC Data Source Administrator is accessible from the start menu or control panel. On 64-bit Windows, to launch the 32-bit ODBC Data Source Administrator you must run `C:\WINDOWS\SysWOW64\odbcad32.exe`. See ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 1 for details.

2. In the ODBC Data Source Administrator, click **System DSN** tab.
3. Scroll through the list of System Data Sources, select **<YourProjectName>DSII** and click **Configure**.
   The Data Source Configuration window opens and displays the data source name, description and the data directory.
4. Now that you have looked at the configuration information for your new driver, click **Cancel** to close the Data Source Configuration window.

Connecting to the Data Source Using Your Customizable Driver

To test your customizable driver using ODBC Test:

1. Start the ODBC Test tool. By default, the ODBC Test application is installed in the following folder: `C:\Program Files (x86)\Microsoft Data Access SDK 2.8\Tools\`.

   Navigate to the folder that corresponds to your driver's architecture (amd64, ia64 or x86) and click `odbcte32.exe` to launch the ANSI version or click `odbct32w.exe` to launch the Unicode version. It is important to run the correct version of the ODBCTest tool for ANSI or Unicode and 32-bit or 64-bit.

2. Attach Visual Studio to the ODBC Test process. To do this, go to Microsoft Visual Studio and then click **Debug** > **Attach to Process**.
3. In the **Attach to Process** window, select **ODBC Test process** and click **Attach**. The process name will be either `odbc32.exe` or `odbct32w.exe`.
4. Add a breakpoint in `Main_Windows.cpp`, on the function `DSIDriverFactory()`. This function runs as soon as the Driver Manager loads the ODBC driver.
5. In the ODBC Test tool, select **Conn** > **Full Connect**.
   The **Full Connect** window opens.
6. Select your Data Source from the list of data sources and click **OK**.
   If you do not see your data source in the list, make sure that you are running the version of the ODBC Test tool that corresponds to the version of the data source that you created. In other words, if you created a 32-bit data source then you should be using the 32-bit version of the ODBC Test tool.
7. You should hit the breakpoint you created and focus should switch to Visual Studio.
8. To continue running the program, select **Debug** > **Continue**.
   The focus returns to the **ODBC Test** window.

Summary – Day One

So far, the following tasks are complete:

- Installing SimbaEngine and building the Quickstart sample driver included with SimbaEngine.
- Learning how to create new driver definitions and Data Source Names in the Windows Registry.
- Viewing DSNs using the ODBC Data Source Administrator.
- Testing drivers using an ODBC-enabled application.
- Setting up a new project for customization based on sample driver projects included in SimbaEngine.

Day Two

Today's goal is to customize your driver, enable logging and establish a connection to your data store. To accomplish this you will visit TODO items 1 to 6.

Remember that when you build the project, you will see the TODO messages in the Output window. To rebuild the whole solution, select **Build** > **Rebuild Solution**. If the Output window is not open, then select **Debug** > **Windows** > **Output**. Double click a TODO message to jump to the relevant section of code.

Construct a Connector Singleton

TODO #1: Construct a Connector Singleton

The `DSIDriverFactory()` implementation in `MainWindow.cpp` is the main entry point that is called from Simba's ODBC layer to create an instance of the DSI implementation. This method is called as soon as the Driver Manager calls `LoadLibrary()` on the ODBC connector.

To construct the connector singleton:

1. Launch Microsoft Visual Studio 2022.
2. Click **File** > **Open** > **Project/Solution**
3. Navigate to `[INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\<YourProjectName>\Source` and then open the `<YourProjectName>_VS201x.vcproj` file.
4. Rebuild your solution and the double click the TODO #1 message section of code. The `Main_Windows.cpp` file opens.
5. Look at the `DSIDriverFactory()` implementation.
6. Specify the location that is used when reading driver settings from the registry. This change is related to rebranding. Replace "Simba" with your company name and change "Quickstart" to the name of your driver in the following line:
   `SimbaSettingReader::SetConfigurationBranding("Simba\\Quickstart");`

   This step, like those in day one, is important to distinguish your driver from our sample and other drivers.
7. Add the processing, if you are building a commercial connector.
8. Click **Save**.

On Linux and UNIX platforms, `DSIDriverFactory()` is implemented in `Main_Unix.cpp`.

Setting Driver Properties

TODO #2: Set the driver properties.

To set driver properties:

1. Double click the TODO #2 message to jump to the relevant section of code. The `QSDriver.cpp` file opens. Look at `SetDriverPropertyValues()` where you will set up the general properties for your driver.
2. Change the DSI_DRIVER_DRIVER_NAME setting. Set this to the name of your driver. (The same name you used to replace "QuickstartDSII" in Day One).
3. Depending on the character sets or Unicode encoding used on your data store, you may want to change the following settings:
   - DSI_DRIVER_STRING_DATA_ENCODING – The encoding of char data within the data store. The default value is ENC_UTF8.
   - DSI_DRIVER_WIDE_STRING_DATA_ENCODING – The encoding of wide character data within the data store. The default is ENC_UTF16_LE.

Setting Logging Details

Here you will customize how the driver logs errors and other information. The important loggers are the driver log for anything not specific to a single connection, and the connection log for anything unique to a single connection or statement within a connection. Following the TODOs below, you can use our provided logger implementation and just rename the output filename. Or you may entirely replace it later with your own implementation of our ILogger interface.

TODO #3: Set the driver-wide logging details.

TODO #4: Set the connection-wide logging details.

To set logging details:

1. Double click the TODO #3 message to jump to the relevant section of code.
2. Change the driver log's file name.
3. Double click the TODO #4 message to jump to the relevant section of code.
4. Change the connection log's file name.
5. Click **Save All.**

By default, the SimbaEngine Quickstart Driver maintains two kinds of log files: one for all driver-based calls and one for each connection created. Update these TODO's if you do not require such fine granularity in logging.

For more information about how to enable logging, refer to the *SimbaEngine Developer Guide*.

Checking Connection Settings

TODO #5: Check Connection Settings.

When the Simba ODBC layer is given a connection string from an ODBC-enabled application, the Simba ODBC layer parses the connection string into key-value pairs. Then, the entries in the connection string and the DSN are sent to the `QSConnection::UpdateConnectionSettings()` function for validation. Validating the correctness (eg. Passwords) is done later in the `Connect()` method.

To verify connection settings:

1. Double click the TODO #5 message to jump to the relevant section of code.
2. The `UpdateConnectionSettings()` function should validate that the key-value pairs in `in_connectionSettings` are sufficient to create a connection. Use the `VerifyRequiredSetting()` or `VerifyOptionalSetting()` utility functions to do this. If any of the values received are invalid, then you should throw an ErrorException seeded with `DIAG_INVALID_AUTH_SPEC`.

For example, the Quickstart driver verifies that the entries within `in_connectionSettings` are sufficient to create a connection by using the following code:

VerifyRequiredSetting(QS_DBF_KEY, in_connectionSettings, out_connectionSettings);

The Quickstart driver requires a single key in the DSN, "DBF" which represents the file location to be searched. This “QS_DBF_KEY” is the connection key to use when looking up the DBF path in the connection string.

If the entries within `in_connectionSettings` are not sufficient to create a connection, then you can ask for additional information from the ODBC-enabled application by specifying the additional, required settings to `out_connectionSettings`. If there are no further entries required, simply leave `out_connectionSettings` empty.

Establishing a Connection

TODO #6: Establish A Connection.

Once `QSConnection::UpdateConnectionSettings()` returns `out_connectionSettings` without any required settings. If there are only optional settings, a connection can still occur, then the Simba ODBC layer calls `QSConnection::Connect()` passing in all the connection settings received from the application.

To establish a connection:

Double click the TODO #6 message to jump to the relevant section of code.
Look at the code that authenticates the user against your data store using the

information provided within the `in_connectionSettings` parameter. The sample code uses the utility functions `GetRequiredSetting()` and `GetOptionalSetting()`. Alternatively, if authentication fails, you can choose to throw an `ErrorException` seeded with `DIAG_INVALID_AUTH_SPEC`. Use the obtained values (eg. hostname, username, password, etc.) to make a connection with your datasource by passing them to your relevant API or network protocol.

You have now authenticated the user against your data store.

Day Three

Today's goal is to return the data used to pass catalog information back to the ODBC-enabled application. Almost all ODBC-enabled applications require the following ODBC catalog functions:

- SQLGetTypeInfo
- SQLTables (CATALOG_ONLY)
- SQLTables (TABLE_TYPE_ONLY)
- SQLTables
- SQLColumns

Creating and Return Metadata Sources

TODO #7: Create and return your Metadata Sources.

`QSDataEngine::MakeNewMetadataTable()` is responsible for creating the sources to be used to return data to the ODBC-enabled application for the various ODBC catalog functions. Each ODBC catalog function is mapped to a unique `DSIMetadataTableId`, which is then mapped to an underlying `MetadataSource` that you will implement and return. Each `MetadataSource` instance is responsible for three things:

1. Creating a data structure that holds the data relevant for your data store: `Constructor`.
2. Navigating the structure on a row-by-row basis: `Move()`.
3. Retrieving data: `GetData()` (See Data Retrieval, Data Retrieval for a brief overview of data retrieval).

Handling DSI_TYPE_INFO_METADATA

SQLGetTypeInfo is used by applications to discover data types supported by your driver. The SDK supports all the types listed below but you may want to modify this metadata source if your tables don't support storing all of them or if some of the default metadata differs from our defaults.

The underlying ODBC catalog function `SQLGetTypeInfo` is handled as follows:

1. When called with `DSI_TYPE_INFO_METADATA`, `QSDataEngine::MakeNewMetadataTable()` will return an instance of `QSTypeInfoMetadataSource()`.
2. The SimbaEngine Quickstart Driver example exposes support for all data types, but due to its underlying file format, it is constrained to support only the following types:

SQL_BIGINT	SQL_BIT	SQL_CHAR
SQL_DECIMAL	SQL_DOUBLE	SQL_INTEGER
SQL_LONGVARCHAR	SQL_LONGWVARCHAR	SQL_NUMERIC
SQL_REAL	SQL_SMALLINT	SQL_TINYINT
SQL_TYPE_DATE	SQL_TYPE_TIME	SQL_TYPE_TIMESTAMP
SQL_VARCHAR	SQL_WCHAR	SQL_WVARCHAR

3. For your driver, you may need to change the types returned and the parameters for the types in `QSTypeInfoMetadataSource::PrepareType()`. You change the passed in SqlTypeInfo object to modify the parameters of the types that are supported.

Handling the other MetadataSources

The other required ODBC catalog functions (including SQLTables (CATALOG_ONLY), SQLTables (TABLE_TYPE_ONLY), SQLTables (SCHEMA_ONLY), SQLTables and SQLColumns) are used by applications to learn about your table structure (ie, their names and contents). These sources are handled as follows:

1. When called with any other `DSIMetadataTableId`, `QSDataEngine::MakeNewMetadataTable()` should return `NULL`. Returning `NULL` will signal SimbaEngine that it should use the metadata helper class returned via `QSDataEngine::CreateMetadataHelper()` along with some default `MetadataSources` to create the data source metadata. You can also choose to return a DSIMetadataSource if you don't want to use the metadata helper.

2. You will need to change:

   - `QSMetadataHelper::QSMetadataHelper()`
     The example constructor retrieves a list of the tables in the data source. You should modify this method to load the tables defined within your data store.

   - `QSMetadataHelper::GetNextTable()`
     In the SimbaEngine Quickstart Driver, this method returns the next table in the data source. You should modify this method to retrieve the next table from your data store.

   - The `DSIExtMetadataHelper` class works by retrieving the identifying information for each table and then opening the table via `QSDataEngine::OpenTable()`. Once you have implemented `QSTable`, the correct metadata will be returned for all of the tables and columns in your data source.

You can now retrieve type metadata from within your data store.

On Linux and UNIX platforms, this metadata is also available using the datatypes command in the iodbctest utility.

Day Four

Today's goal is to enable data retrieval from within the driver. We will discuss:

- Opening a table defined within your data store
- Retrieving the column information for the table
- Retrieving data

Enabling Data Retrieval

TODO #8: Open A Table.

`QSDataEngine::OpenTable()` is the entry point where Simba SQL Engine requests that tables involved in the query be opened.

`QSTable` is an implementation of `DSIExtSimpleResultSet`, an abstract class provided by Simba that provides for basic forward-only result set traversal. The main role of `QSTable` is to translate the stored data from your native data format into SQL Data types.

The Quickstart sample driver is implemented for Tabbed Unicode Files. The sample driver translates the text from UTF16-LE strings into the SQL Data types defined for each column within the configuration dialog.

In the Quickstart driver, `QSTable` uses a `TabbedUnicodeFileReader`, which provides an interface to navigate between lines within a Unicode text file. This class preprocesses each row in the file to determine the starting file offset of each column in the row. Its GetData method takes a `columnIndex` and uses it to calculate the exact position in the file where the column's data resides. The method repositions the file and retrieves the data as if from a byte-buffer. See *Appendix C*: Data Retrieval for a brief overview of data retrieval.

To enable data retrieval:

1. Modify the `QSDataEngine::OpenTable` method to check that the supplied catalog, schema and table names are valid and correspond to a table defined in your data store. If they are not, you should return null to indicate that the table does not exist. If the inputs are valid, then a new instance of `QSTable` is returned.
2. Make the following changes to `QSTable` for it to work with your data store:
   a. To return the catalog, schema and table names for your table:
      - `QSTable::QSTable()`: The constructor must be modified to take in the catalog, schema and table names and save them in member variables.
      - `QSTable::GetCatalogName()`: Returns `QS_CATALOG`;
      - `QSTable::GetSchemaName()`: Returns `simba_wstring()` (because it does not

support schemas);

- `QSTable::GetTableName()`: Returns `m_tableName`;

b. To return the columns defined for your table:

- `QSTable::InitializeColumns()`: Modify the method so that, for each column defined in the table, you define a `DSIResultSetColumn` in terms of SQL types. Here is an example of pseudo code for the new method:

- AutoPtr<DSIResultSetColumns> columns;

  Get all column information from your data store for the table

  For Each Defined Column

  {

  AutoPtr<DSIColumnMetadata> columnMetadata(

  new DSIColumnMetadata());

  columnMetadata->m_catalogName = m_catalogName;

  columnMetadata->m_schemaName = m_schemaName;

  columnMetadata->m_tableName = m_tableName;

  columnMetadata->m_name = //column name

  columnMetadata->m_label = //localized column name

  columnMetadata->m_unnamed = false;

  columnMetadata->m_charOrBinarySize = //the length in

  bytes

  columnMetadata->m_nullable = DSI_NULLABLE;

  // Change the first parameter of this method to the SQL

  // type that maps to your data store type.

  SqlTypeMetadata* sqlTypeMetadata =

  SqlTypeMetadataFactory::MakeNewSqlTypeMetadata(

  SQL_WVARCHAR, TDW_BUFFER_OWNED);

  columns->AddColumn(

  new DSIResultSetColumn(

  sqlTypeMetadata,

  columnMetadata.Detach()));

```
}


m_columns.Attach(columns.Detach());
```

c. Modify the following three methods. The methods are responsible for navigating a data structure containing information about one table in the data store and retrieving data from the table:

- `QSTable::MoveToBeforeFirstRow()`
- `QSTable::MoveToNextRow()`
- `QSTable::GetData()`

In your class:

- Implementing a streaming interface for the data in the table within your data store is best.
- Provide the ability to navigate forward from one table row to the next.
- Provide the ability to navigate across columns within the row.
- Provide the ability to read the data associated with the current row and column combination.

d. Modify `QSTable::DoCloseCursor()`. This is a callback method called from Simba SQL Engine to indicate that data retrieval has completed and that you may now do any tasks related to closing the connection to your data store.

You can now retrieve data and see the rest of the metadata from your data store. You should be able to:

- Run `SQLTables()` and `SQLColumns`() from within `ODBCTest32.exe` (Unicode) and see the correct metadata returned.
- Execute queries from any ODBC-enabled application such as Microsoft Excel, Microsoft Access, Microsoft SQL Server or Crystal Reports and see the results returned from your data store.

On Linux and UNIX platforms, lists of catalogs, schemas, tables and types are available using the qualifiers, owners, tables and types commands in the iodbctest utility.

Day Five

Today's goal is to start productizing your driver. Additionally, you can also start localizing your driver error messages. Refer to SimbaEngine Developer Guide for more details.

Configuring Error Messages

TODO #9: Register Messages xml file for handling by DSIMessageSource.

All the error messages used within your DSI implementation are stored in a file called `QSMessages.xml`

To configure error messages:

1. Rename the `QSMessages.xml` file to something appropriate to your data store.
2. Double click the TODO #9 message to jump to the relevant section of code.
3. Update the line associated with the TODO to match the new name of the file.
4. Open the `QSMessages.xml` file and change all instances of the following items:
   - The letters "`QS`" to a two letter abbreviation of your choice
   - The word "`Quickstart`" to a name relating to your driver
5. When you are done, you should revisit each exception thrown within your DSI implementation and change the parameters to match as well. This will rebrand your converted SimbaEngine Quickstart Driver for your organization.

TODO #10: Set the vendor name, which will be prepended to error messages.

The vendor name is prepended to all error messages that are visible to applications. The default vendor name is Simba.

To set the vendor name:

1. Double click the TODO #10 message to jump to the relevant section of code.
2. Set the vendor name as shown in the commented code.

Adding Finishing Touches

You have now completed all TODO's in the project. However, there are still a couple of final steps before you have a fully functioning driver.

To add finishing touches:

1. Rename all files and classes in the project to have the two-letter abbreviation you chose as part of TODO #9.

2. Create a driver configuration dialog. This dialog is presented to the user when they use the ODBC Data Source Administrator to create a new ODBC DSN or configure an existing one. The C++ SimbaEngine Quickstart Driver project contains an example ODBC configuration dialog that you can look at, as an example. You can find the source under the Setup folder within the SimbaEngine Quickstart Driver project.

3. To see the driver configuration dialog that you created, run the ODBC Data Source Administrator, open the Control Panel, select Administrative Tools, and then select Data Sources (ODBC). If your Control Panel is set to view by category, then Administrative Tools is located under System and Security.

> **Important:** If you are using 64-bit Windows with 32-bit applications, you must use the 32-bit ODBC Data Source Administrator. You cannot access the 32-bit ODBC Data Source Administrator from the start menu or control panel in 64-bit Windows. Only the 64-bit ODBC Data Source Administrator is accessible from the start menu or control panel. On 64-bit Windows, to launch the 32-bit ODBC Data Source Administrator you must run `C:\WINDOWS\SysWOW64\odbcad32.exe`. See ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 1 for details.

On Linux and UNIX platforms, creating a driver configuration dialog is possible. However, the Quickstart sample driver for Linux and UNIX platforms does not include a configuration dialog.

Conclusion

By modifying the Quickstart sample driver included with SimbaEngine, you have created your own read-only ODBC driver that connects to your data store.

Reference

This section contains more information that you may find useful when developing your sample ODBC driver.

Appendex A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit

On a 64-bit Windows system, you can execute 64-bit and 32-bit applications transparently. Currently, many applications are available in 32-bit versions only. Running 32-bit applications on 64-bit operating systems is common.

Note: Microsoft Excel is available in 32-bit and 64-bit versions.

In a single running process, all of the code must be either 64-bit or 32-bit. 64-bit applications can only load 64-bit ODBC drivers and 32-bit applications can only load 32-bit ODBC drivers.

On a 64-bit Windows system, the ODBC Data Source Administrator that you access through the Control Panel by default configures data sources for 64-bit applications. However, you must use the 32-bit version of the ODBC Data Source Administrator to configure data sources for 32-bit applications.

PROBLEM: You cannot access the 32-bit ODBC Data Source Administrator from the Start menu or Control Panel in 64-bit Windows.

SOLUTION: To create new 32-bit data sources or modify existing ones on 64-bit Windows you must run C:\WINDOWS\SysWOW64\odbcad32.exe. Create a shortcut to the 32-bit ODBC Data Source Administrator on your Desktop or Start menu if you configure 32-bit data sources frequently.

Appendex B: Bitness and the Windows Registry

A 64-bit application cannot use a 32-bit ODBC driver, and vice versa. In the Registry on computers running 64-bit Windows operating systems, system-wide information about 64-bit ODBC drivers is stored in the HKEY_LOCAL_MACHINE/SOFTWARE/ODBC subkey and system-wide information about 32-bit ODBC drivers is stored in the HKEY_LOCAL_ MACHINE/ SOFTWARE/WOW6432NODE/ODBC subkey.

For 32-bit applications running on a 64-bit computer, 32-bit data sources appear the same as if the application was running on a 32-bit computer.

On computers running 32-bit Windows operating systems, system-wide information about ODBC drivers is stored in the HKEY_LOCAL_MACHINE/SOFTWARE/ODBC subkey. 32-bit Windows operating systems cannot run 64-bit applications or drivers.

32- and 64-bit ODBC Data Source Administrators discussed in Appendex A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit use information about drivers stored in the respective Windows Registry keys.

The ODBCINST.INI Subkey

To define a name and set the location of drivers, the ODBC Data Source Administrator matching the bitness of the operating system uses definitions in the subkey HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI and the 32-bit ODBC Data Source Administrator on 64-bit Windows uses Registry keys created in HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBCINST.INI

The ODBCINST.INI subkey contains a subkey for each driver. The subkey for each driver includes the following string values:

- **Driver** provides the path to the driver DLL
- **Setup** provides the location of the driver setup DLL
- **Description** briefly describes the driver

The ODBCINST.INI subkey also contains a subkey named ODBC Drivers that contains string values to match the name of each driver subkey. Each string value is the text Installed

For more details on information in the Windows Registry contained in the ODBCINST.INI subkey, see the MSDN topic Registry Entries for ODBC Components at https://msdn.microsoft.com/en-us/library/ms714818%28v=vs.85%29.aspx.

The ODBC.INI Subkey

To connect a driver to a database, the ODBC Data Source Administrator matching the bitness of the operating system uses definitions in the subkey HKEY_LOCAL_MACHINE/SOFTWARE/ ODBC/ODBC.INI and the 32-bit ODBC Data Source Administrator on 64-bit Windows uses Registry keys created in HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ ODBC.INI

The ODBC.INI subkey contains a subkey for each Data Source Name (DSN). The subkey for each DSN includes the following string values:

- **Driver** defines the name of the driver to use for connecting to the database, as set in the ODBCINST.INI subkey.

The Driver value can also set a file path to the driver DLL.

- **DBF** defines the path to the database to which the DSN creates a connection.
- **Description** briefly describes the DSN.

The ODBC.INI subkey also contains a subkey named ODBC Data Sources that contains string values to match the name of each DSN subkey. Each string value is the name of the driver that the DSN uses.

For more details on information in the Windows Registry contained in the ODBC.INI subkey, see the MSDN topic Registry Entries for Data Sources at https://msdn.microsoft.com/en-us/library/ms712603%28v=vs.85%29.aspx.

Appendix C: Data Retrieval

In the Data Store Interface (DSI), the following two methods actually perform the task of retrieving data from your data store:

- Each `MetadataSource` implementation of `GetMetadata()`
- `QSTable::GetData()`

Both methods will provide a way to uniquely identify a column within the current row. For `MetadataSource`, Simba SQL Engine will pass in a unique column tag (see `DSIOutputMetadataColumnTag`). For `QSTable`, Simba SQL Engine will pass in the column index.

In addition, both methods accept the following three parameters:

1. `in_data`

   The SQLData into which you must copy the value of your cell. This class is a wrapper around a buffer managed by the Simba SQL Engine. To access the buffer, you call its GetBuffer() method. The data you copy into the buffer must be formatted as a SQL Type (see http://msdn.microsoft.com/en-us/library/ms710150%28VS.85%29.aspx for a list of data types and definitions). Therefore, if your data is not stored as SQL Types, you will need to write code to convert from your native format.

   The type of this parameter is governed by the metadata for the column that is returned by the class. Thus, if you set the SQL Type of column 1 in `QSTable::InitializeColumns()` to `SQL_INTEGER`, then when `QSTable::GetData()` is called for column 1, you will be passed a `SQLData` that wraps an `int` datatype. For `MetadataSource`, the type is associated with the column tag (see `DSIOutputMetadataColumnTag.h`). For character or binary data you must call `SetLength()` before calling `GetBuffer()`. Not doing so may result in a heap-violation. See `QSTypeUtilities.h` for an example on how to handle character or binary data.

2. `in_offset`

   Character, wide character and binary data types can be retrieved in parts. This value specifies where, in the current column, the value should be copied from. The value is usually 0.

3. `in_maxSize`

   The maximum size (in bytes) that can be copied into the `in_data` parameter. For character or binary data, copying data that is greater than this size can result in a data truncation warning or a heap-violation.

SqlData types

SqlData objects represent the SQL types and encapsulate the data in a buffer. When you have a SqlData object and would like to know what data type it is representing, use `GetMetadata()->GetSqlType()` to see what the associated `SQL_* type` is.

For information how SQL types map to C++ types, see Appendix G in the *SimbaEngine Developer Guide*.

Fixed length types

The structures used to store the fixed-length data types represented by SqlData objects are:

SQL_BIT

SQL_DATE

SQL_DECIMAL

SQL_DOUBLE

SQL_FLOAT

SQL_INTERVAL_DAY

SQL_INTERVAL_DAY_TO_HOUR

SQL_INTERVAL_DAY_TO_MINUTE

SQL_INTERVAL_DAY_TO_SECOND

SQL_INTERVAL_HOUR

SQL_INTERVAL_HOUR_TO_MINUTE

SQL_INTERVAL_HOUR_TO_SECOND

SQL_INTERVAL_MINUTE

SQL_INTERVAL_MINUTE_TO_SECOND

SQL_INTERVAL_MONTH

SQL_INTERVAL_SECOND

SQL_INTERVAL_YEAR

SQL_INTERVAL_YEAR_TO_MONTH

SQL_NUMERIC

SQL_REAL

SQL_BIGINT

SQL_INTEGER

SQL_SMALLINT

SQL_TINYINT

SQL_TYPE_DATE

SQL_TYPE_TIME

SQL_TYPE_TIMESTAMP

More information on Date, Time and DateTime types

The associated SQL types for date, time, and datetime are `SQL_TYPE_DATE`, `SQL_TYPE_TIME`, and `SQL_TYPE_TIMESTAMP`. Please note that the `SQL_TIME`, `SQL_DATE`, and `SQL_TIMESTAMP` are ODBC 2.x types while the `SQL_TYPE_*` types are ODBC 3.x types, so you should be sure to use the `SQL_TYPE_*` types since you are developing an ODBC 3.x driver.

Simple Fixed-Length Data Example

For a SQL_INTEGER, the SQLData will contain a simba_int32 which you must copy your integer value into. The example below illustrates how this might be achieved.

```
switch (in_data->GetMetadata()->GetSqlType())
{
 case SQL_INTEGER:
 {
  simba_int32 value = 1234;
  *reinterpret_cast<simba_int32*>(in_data->GetBuffer()) = value;
 }
}
```

Variable Length Types

The following variable-length data types are stored in buffers and represented by SqlData objects:

SQL_BINARY

SQL_CHAR

SQL_LONGVARBINARY

SQL_LONGVARCHAR

SQL_VARBINARY

SQL_VARCHAR

SQL_WCHAR

SQL_WLONGVARCHAR

SQL_WVARCHAR

You may find that the `DSITypeUtilities::OutputWVarCharStringData` and `OutputVarCharStringData` are useful for setting character data.

Simple Variable-Length Data Example

The `SQL_CHAR` example below illustrates how the type utilities might be used while the `SQL_VARCHAR` example shows a simple example using memcpy. In practise, `SQL_CHAR`, `SQL_`

`VARCHAR` and `SQL_LONGVARCHAR` will not need separate cases to handle them and there will also be other considerations such as having to deal with offsets into the data.

```
switch (in_data->GetMetadata()->GetSqlType())
{
 case SQL_CHAR:
 {
  simba_string stdString("Hello");
  return DSITypeUtilities::OutputVarCharStringData(
     &stdString,
     in_data,
     in_offset,
     in_maxSize);
 }
 case SQL_VARCHAR:
 {
     simba_string stdString("Hello");
     simba_uint32 size = stdString.size();
     in_data->SetLength(size);
     memcpy(in_data->GetBuffer(), stdString, size);
return false;
 }
}
```

Data Conversion in Practice

In the SimbaEngine Quickstart example, when `GetData()` is called the values are read from the tabbed Unicode file (in `TabbedUnicodeFileReader::GetData`), converted to simba_wstrings (in `QSTable::ReadWholeColumnAsString`) and then converted to the requested SQL data type (in `QSTable::ConvertData`). This works well because the data source is a text file and a good cross-platform example.

For your data source, if you're already getting data of the correct type—integers, for example—then ideally you should skip the conversion to strings so you can achieve better performance. Be aware of which data types map to which SQL Types, as well as how to represent them in the expected format. Then you can set the buffer in an appropriate manner.

NULL Values

To represent a null value, directly set the SqlData object as null:

```
in_data->SetNull(true);
```

Appendix D: C++ Server Configuration

To establish a connection, the connection settings for the driver are normally retrieved directly from the ODBC DSN. However, when the driver is a server, the settings cannot be retrieved directly because the DSN refers to the client instead of a specific driver. In addition, there would also be security concerns, if a given client has control over server-specific settings. Therefore, to establish a connection when a driver is a server, the connection settings need to be augmented.

The information in this section only applies if you are using 32-Bit Windows. If you are using 64-bit Windows (with either 32-bit or 64-bit applications), the file paths must be configured appropriately. Please see Windows Registry 32-Bit vs. 64-Bit Windows Registry 32-Bit vs. 64-Bit on page 1 for details.

For the UltraLight sample driver, the registry entries under `HKEY_LOCAL_MACHINE/SOFTWARE/SIMBA/ULTRALIGHT/SERVER` are used to enable this server-specific behavior. The settings augment the connection settings that are passed in during a connection.

On Linux and UNIX platforms, the configuration entries are located in the `.simbaserver.ultralight.ini` file.

To set the UltraLight sample driver up as a server, build the UltraLight solution using a server configuration (i.e. Debug_Server or Release_Server). This will build the server executable.

The rest of the server settings are located under sub-nodes of `HKEY_LOCAL_MACHINE/SOFTWARE/SIMBA/ULTRALIGHT/SERVER`. For full list of possible server configuration parameters, please see the SimbaClientServer User Guide.

On Linux and UNIX platforms, to set the UltraLight sample driver up as a server you need to:

1. Build UltraLight using the debug (or release) server configuration:
   ```
   BUILDSERVER=exe make -f UltraLight.mak debug
   ```
2. Configure the server as required in the other sections of the `.simbaserver.ultralight.ini` file.

For further details on setting up a connection between a client and server, please see the *SimbaClientServer* User Guide. Once you have configured the client and server, you should be able to connect to your data source.

Appendix E: C++ Server Configuration

To establish a connection, the connection settings for the driver are normally retrieved directly from the ODBC DSN. However, when the driver is a server, the settings cannot be retrieved directly because the DSN refers to the client instead of a specific driver. In addition, there would also be security concerns, if a given client has control over server-specific settings. Therefore, to establish a connection when a driver is a server, the connection settings need to be augmented.

Important: Windows Registry keys listed below apply to 32-Bit Windows and 64-bit Windows when running 64-bit applications. If you are using 64-bit Windows with 32-bit applications, then adjust the paths as described in Appendex B: Bitness and the Windows Registry .

For the Quickstart sample driver, the registry entries under `HKEY_LOCAL_MACHINE/SOFTWARE/SIMBA/QUICKSTART/SERVER` are used to enable this server-specific behavior. The settings augment the connection settings that are passed in during a connection.

On Linux and UNIX platforms, the configuration entries are located in the .simbaserver.quickstart.ini file.

To set up the Quickstart sample driver as a server:

1. Build the Quickstart solution using a server configuration (i.e. Debug_Server or Release_Server). This will build the server executable.
2. In the Windows Registry, navigate to the key `HKEY_LOCAL_MACHINE/SOFTWARE/SIMBA/QUICKSTART/SERVER`, and then add the following string value: `DBF=[INSTALL_DIRECTORY]\Examples\Databases\Quickstart`.
3. Additonal server settings are located under sub-nodes of `HKEY_LOCAL_MACHINE/SOFTWARE/SIMBA/QUICKSTART/SERVER`. For full list of possible server configuration parameters, see the *SimbaClientServer* User Guide.

On Linux and UNIX platforms, to set the Quickstart sample driver up as a server you need to:

1. Build Quickstart using the debug (or release) server configuration:
   `BUILDSERVER=exe make -f Quickstart.mak debug`
2. Add the DBF value to the [Server] section of the .simbaserver.quickstart.ini file:
   `DBF=[INSTALL_DIRECTORY]/Examples/Databases/Text`
3. Configure the server as required in the other sections of the .simbaserver.quickstart.ini file.

For further details on setting up a connection between a client and server, please see the *SimbaClientServer* User Guide. After configuring the client and server, you should be able to connect to your data source.

Third Party Licenses

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2014 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

OpenSSL License

Copyright (c) 1998-2011 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)"

4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.

5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.

6. Redistributions of any form whatsoever must retain the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young(eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay License

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)

All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are aheared to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

"This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"

The word 'cryptographic' can be left out if the rouines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:

"This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Expat License

"Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ""Software""), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ""AS IS"", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NOINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

Stringencoders License

Copyright 2005, 2006, 2007

Nick Galbreath -- nickg [at] modp [dot] com

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the modp.com nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This is the standard "new" BSD license:

http://www.opensource.org/licenses/bsd-license.php

dtoa License

The author of this software is David M. Gay.

Copyright (c) 1991, 2000, 2001 by Lucent Technologies.

Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software.

THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.