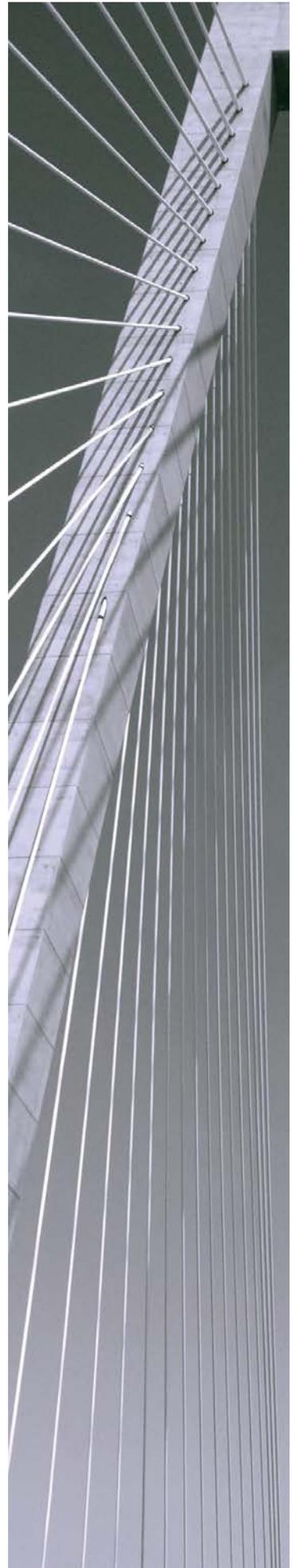# SimbaEngine X version 10.1

Build a C# ODBC Driver in 5 Days

Last Revised: November 2016

Simba Technologies Inc.

Copyright © 2015 Simba Technologies Inc. All Rights Reserved.

Trademarks

Simba, the Simba logo, SimbaEngine, and Simba Technologies are registered trademarks of Simba Technologies Inc. in Canada, United States and/or other countries. All other trademarks and/or servicemarks are the property of their respective owners.

Kerberos is a trademark of the Massachusetts Institute of Technology (MIT).
Linux is the registered trademark of Linus Torvalds in Canada, United States and/or other countries.

Mac, Mac OS, and OS X are trademarks or registered trademarks of Apple, Inc. or its subsidiaries in Canada, United States and/or other countries.

Microsoft SQL Server, SQL Server, Microsoft, MSDN, Windows, Windows Azure, Windows Server, Windows Vista, and the Windows start button are trademarks or registered trademarks of Microsoft Corporation or its subsidiaries in Canada, United States and/or other countries.

Red Hat, Red Hat Enterprise Linux, and CentOS are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in Canada, United States and/or other countries.

Solaris is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

SUSE is a trademark or registered trademark of SUSE LLC or its subsidiaries in Canada, United States and/or other countries.

Ubuntu is a trademark or registered trademark of Canonical Ltd. or its subsidiaries in Canada, United States and/or other countries.

All other trademarks are trademarks of their respective owners.

Contact Us

Simba Technologies Inc.
938 West 8th Avenue
Vancouver, BC, Canada
V5Z 1E5

www.simba.com

Telephone +1 (604) 633-0008   sales: extension 2, support: extension 3

Fax +1 (604) 633-0004

Information and product sales: solutions@simba.com

Technical support: support@simba.com

Follow us on Twitter: @simbatech

Table of Contents

Introduction

This guide will show you how to create your own, custom ODBC driver using SimbaEngine. It will walk you through the steps to modify and customize the included DotNetQuickstart sample driver. At the end of five days, you will have a read-only driver that connects to your data store.

About SimbaEngine

SimbaEngine is a complete implementation of the ODBC 3.80 specification, which provides a standard interface to which any ODBC enabled application can connect. ODBC is one the most established and widely supported APIs for connecting to and working with databases. At the heart of the technology is the ODBC driver, which connects an application to the database. For more information about ODBC, see http://www.simba.com/odbc.htm. For complete information on the ODBC specification, see the MSDN ODBC Programmer's Reference, available from the Microsoft web site at http://msdn.microsoft.com/en-us/library/ms714562(VS.85).aspx

The libraries of SimbaEngine hide the complexity of error checking, session management, data conversions and other low-level implementation details. They expose a simple API, called the Data Store Interface API or DSI API, which defines the operations needed to access a data store. Full documentation for SimbaEngine is available on the Simba website at http://www.simba.com/odbc-sdk-documents.htm.

You use SimbaEngine to create a file that will be accessed by common reporting applications and to access your data store when SimbaEngine executes an SQL statement. You create a custom-designed DSI implementation (DSII) that connects directly to your data source. Then, you create the executable by linking libraries from SimbaEngine with the DSI implementation that you have written. In the process, the project files or make files will 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.

About the DotNetQuickstart sample driver

The DotNetQuickstart driver is a sample DSI implementation of an ODBC driver, written in C#, which reads files that are in tabbed Unicode text format. Because text files are not a SQL-aware data source, the Simba SQLEngine component must be used to perform the necessary SQL processing.

The DotNetQuickstart driver helps you to prototype a DSI implementation for your own data store so you can learn how SimbaEngine works. You can also use it as the foundation for your commercial DSI implementation if you are careful to remove the shortcuts and simplifications that it contains. This is a fast and effective way to get a data access solution to your customers.

A typical design pattern for a DSI implementation is shown in the following UML diagram.
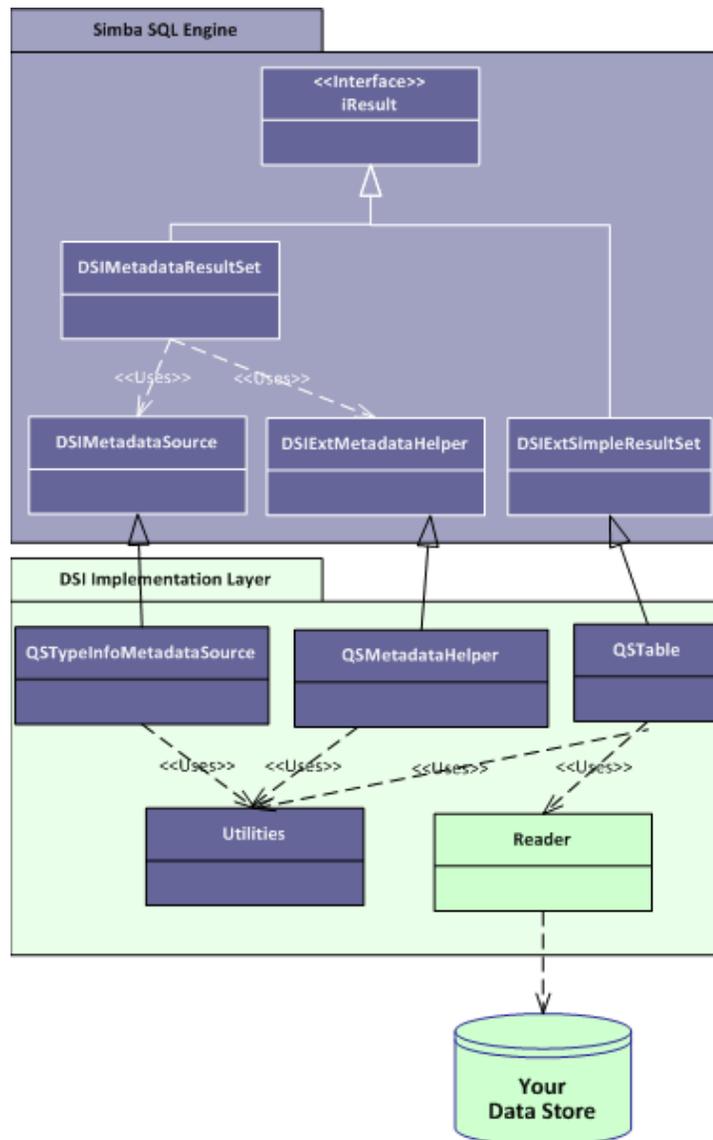
Figure 1: Design pattern for a DSI implementation.

There is a 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 and the QSUtilities class contains a collection of utility functions that are used by the DSI.

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

To add metadata access, the QSTypeInfoMetadataSource class is implemented and the DSIExtMetadataHelper class is used. The QSTypeInfoMetadataSource class is responsible for generating the metadata for SQLGetTypeInfo with the supported data types. The DSIExtMetadataHelper class is responsible for iterating through tables and stored procedures so the engine can generate catalog function metadata.

Note	Collaborative Query Execution (CQE) is an optimization feature that lets you divide SQL queries between the Simba SQL engine and your own SQL engine. The DotNetQuickstart sample driver does not use CQE, and cannot be modified to use CQE, because the `DotNetDSIExt` library in SimbaEngine does not expose CQE methods or classes.

Overview

The series of steps to take to get a prototype DSI implementation working with your data store is as follows:

- Set up the development environment
- Make a connection to the data store
- Retrieve metadata
- Work with columns
- Retrieve data

In the DotNetQuickstart driver, the areas of the code that you need to change are marked with "TODO" messages along with a short explanatory message. Most of the areas of the code that you need to modify are for productization rather than actually connecting your data store to Simba SQLEngine. These are things like naming the driver, setting the properties that configure the driver, and naming the log files. The other areas of the code that you will modify are related to getting the data and metadata from your data store into the Simba SQLEngine. Since the DotNetQuickstart driver already has the classes and code to do this against the example data store, all you have to do is modify the existing code to make your driver work against your own data store.

Day One

Today's task is to set up the development environment and project files for your driver. By the end of the day, you will have compiled and tested your first ODBC driver.

Install SimbaEngine

Note: If you have a previous version of SimbaEngine installed, uninstall it before installing the new one.

1.   If Visual Studio is running, close it.
2.   Run the SimbaEngine setup executable that corresponds to your version of Visual Studio and follow the installer's instructions.

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.

Build the DotNetQuickstart example driver

Note: Visual Studio 2013 is used for the examples, but Visual Studio 2015 is also supported.

1.   Launch Microsoft Visual Studio.
2.   Click **File** > **Open** > **Project/Solution**.
3.   Navigate to
     `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\DotNetQuickstart\`
     `Source` and then open the `DotNetQuickstart_VS2013.sln` file.
     The default `[INSTALL_DIRECTORY]` is `C:\Simba Technologies`.

     The solution contains two projects: Quickstart, which is the C# driver and QuickstartCLIDSI, which is the driver's native component (C++ CLI). Every .NET ODBC driver built using SimbaEngine will have two components, one managed and one native. The native component (in this case QuickstartCLIDSI) has only one function, to create an instance of the .NET driver object.

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.

Install the assembly into the Global Assembly Cache

Each time you build the DLL, it must be installed to the Global Assembly Cache (GAC) before it can be used. To run the Global Assembly Cache tool, use the Visual Studio Command Prompt. You must run this command as an administrator.

1.   On the taskbar, click **Start** > **All Programs** > **Microsoft Visual Studio** > **Visual Studio Tools**.

2. Right-click **Visual Studio Command Prompt** and select "Run as administrator".

3. Change to the directory that contains the DLL file. For 32-bit drivers, type a command that is similar the following:

```
cd
[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\DotNetQuickstart\Bin\
Windows_vs2013\debug32md
```

For 64-bit drivers, type:

```
cd
[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\DotNetQuickstart\Bin\
Windows_vs2013\debug64md
```

4. Type the following command to install the assembly into the GAC:

```
gacutil.exe /i DotNetQuickstartDSII.dll
gacutil.exe /i QuickstartConfigDialog.dll
```

You will see the message, "Assembly successfully added to the cache" if the operation was successful.

5. In addition to the DLL of your driver, other DLLS need to be installed in the GAC. These files were installed in the GAC during SIMBAENGINE installation. In order to check for these assemblies in the GAC run the following commands:

```
gacutil.exe /l Simba.DotNetDSI

gacutil.exe /l Simba.DotNetDSIExt
```

Note: If an assembly is already installed in the GAC, then it must be uninstalled from GAC before installing it again. To remove an assembly, run the following command (as administrator):

```
gacutil.exe /u <assembly_display_name>
```

Examine the registry keys added by the SimbaEngine installer

The SimbaEngine installer automatically added or updated the following registry keys that define Data Source Names (DSNs) and driver locations:

- **ODBC Data Sources** - lists each DSN/driver pair

- **DotNetQuickstartDSII** - defines the Data Source Name (DSN). Used by the ODBC Driver Manager to connect your driver to your database.

- **ODBC Drivers** - lists the drivers that are installed

- **DotNetQuickstartDSIIDriver** - defines the driver and its setup location. The ODBC Driver Manager uses this key.

To view the registry keys, do the following:

1. Run `regedit.exe.`

2. To view the registry keys that are related to Data Source Names, expand the folders in the Registry Editor to the following location:

    For 32-bit drivers on 32-bit Windows and 64-bit drivers on 64-bit Windows:

    ```
    HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBC.INI
    ```

    For 32-bit drivers on 64-bit Windows:

    ```
    HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBC.INI
    ```

3. To view the registry keys that are related to ODBC drivers, expand the folders in the Registry Editor to the following location:

    For 32-bit drivers on 32-bit Windows and 64-bit drivers on 64-bit Windows:

    ```
    HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI
    ```

    For 32-bit drivers on 64-bit Windows:

    ```
    HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBCINST.INI
    ```

Your custom driver installer will eventually have to create similar registry keys.

Note: Registry keys for 32-bit and 64-bit ODBC drivers are installed in different areas of the Windows registry. See Appendix B: Windows Registry 32-Bit vs. 64-Bit on page 24 for more information.

View the data source 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 (other than Windows 8), 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 `%WINDIR%\SysWOW64\odbcad32.exe`. See Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 23 for details.

2. In the ODBC Data Source Administrator, click the **System DSN** tab.

3. Scroll through the list of System Data Sources, select **DotNetQuickstartDSII** and then 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 the driver, click **Cancel** to close the Data Source Configuration window.

Test the data source

To test the data source that we have created, you can use any ODBC application, such as, for example, Microsoft Excel, Microsoft Access or ODBCTest. In this section, we will use the ODBC Test tool, which is available in the Microsoft Data Access (MDAC) 2.8 Software Development Kit (SDK). To download SimbaEngine, visit the following Microsoft Web site:
http://www.microsoft.com/downloads/details.aspx?FamilyID=5067faf8-0db4-429a-b502-de4329c8c850&displaylang=en

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 then click `odbcte32.exe` to launch the ANSI version or click `odbct32w.exe` to launch the Unicode version.

    Note: 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, select **Conn > Full Connect**.
    The Full Connect window opens.

3. Select the **DotNetQuickstartDSII** Data Source from the list of data sources and then 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.

4. When the tool connects to the data source, you will see the message, "Successfully connected to DSN 'DotNetQuickstartDSII'".

Set up a new project to build your own ODBC driver

Now that you have built the example driver, you are ready to set up a development project to build your own ODBC driver.

Note: It is very important that you create your own project directory. You might be tempted to just modify the sample project files but we strongly recommend against this, because when you install a new release of SimbaEngine, changes you make will be lost and there may be times, for debugging purposes, that you will need to see if the same error occurs using the sample drivers. If you have modified the sample drivers, this will not be possible.

To rename the projects:

1. In your Windows Explorer window, copy the
   `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\DotNetQuickstart`
   directory and paste it to the same location. This will create a new directory called
   "`DotNetQuickstart - 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 the `Source` directory and then right-click the `DotNetQuickstart_VS2013.sln` file.

3. Select **Open with > Microsoft Visual Studio Version Selector**.

4. In the Microsoft Visual Studio menu, click **View > Solution Explorer**.

5. Using the Solution Explorer, rename the following items:

   - Rename **DotNetQuickstart_VS2013** solution to **<YourProjectName>_VS2013**.

   - Rename the C# project **Quickstart_VS2013** to **<YourProjectName>_VS2013**.

   - Rename **QuickstartCLIDSI** to **<YourProjectName>CLIDSI**.

   - Rename **QuickstartConfigDialog_VS2013** to
     <YourProjectName>ConfigDialog_VS2013.

   - Rename **QuickstartConfigDSN** to **<YourProjectName>ConfigDSN**.

   For example, if your project name was AceData, your solution might look like this:
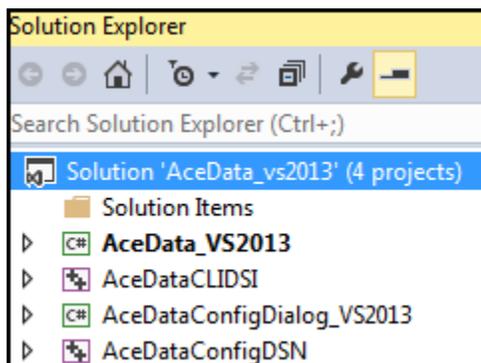
6. Right-click **<YourProjectName>_VS2013** and select **Properties**.

7. In the Assembly name text box, replace **DotNetQuickstartDSII** with **<YourProjectName>DSII**.

8. Click **File > Save All**.

Your project is renamed but you still need to update the namespaces.

To update the namespaces:

To update the namespaces you must remove and re-add the references, because the references are not automatically updated when the projects are renamed.

1. In the solution explorer, right-click on **<YourProjectName>_CLIDSI,** and select **Properties**.

2. Select **Common Properties** > **References**, select **<YourProjectName>_VS2013**, then select **Remove Reference**.

3. Select **Add New Reference**, select the reference you just removed, (**<YourProjectName>_VS2013)**, and select **OK**. For example, if your project was named AceData, you would do the following steps:
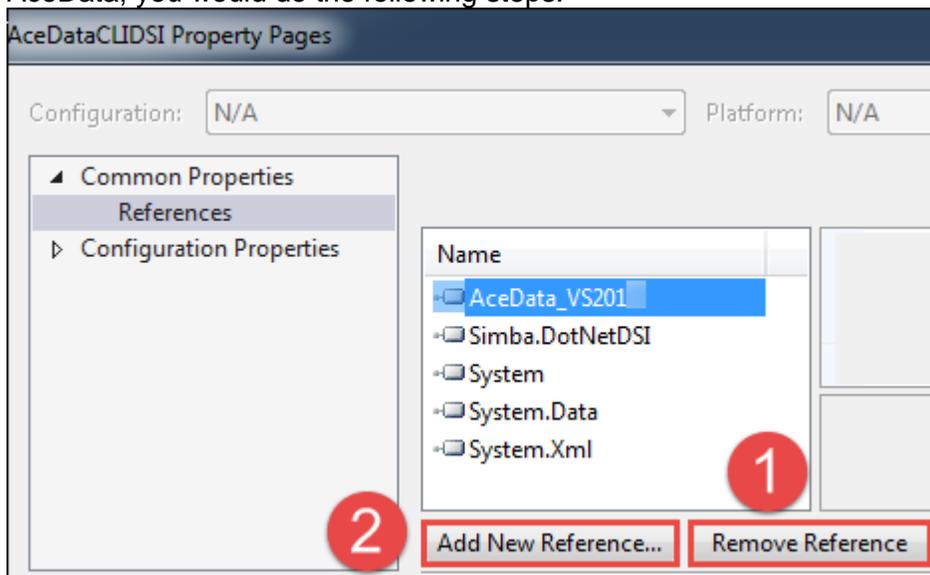
4. Select **OK** to return to the Solution Explorer.

5. For the **<YourProjectName>ConfigDSN** project, repeat the above procedure to remove and re-add the reference to the **<YourProjectName>ConfigDialog_VS2013** project.

6. Select **File** > **Save All**.

You can now rebuild your renamed project.

Build your new driver

1. Click **Build** > **Configuration Manager** and make sure that the active solution configuration is "Debug_MTDLL" and then click **Close**.

2. Click **Build** > **Build Solution** or press F7 to build the driver.

Update the Global Assembly Cache

Each time you build the DLL, it must be installed to the Global Assembly Cache (GAC).

1. On the taskbar, click **Start** > **All Programs** > **Microsoft Visual Studio** > **Visual Studio Tools**.

2. Right-click **Visual Studio Command Prompt** and select "Run as administrator".

3. Change to the directory that contains the DLL file. For 32-bit drivers, type a command that is similar the following:

```
cd [INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\<your project
directory>\Bin\Windows_vs2013\debug32md
```

For 64-bit drivers, type:

```
cd [INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\<your project
directory>\Bin\Windows_vs2013\debug64md
```

4. Type the following commands to install your assemblies into the GAC:

```
gacutil.exe /i <YourProjectName>DSII.dll
```
```
gacutil.exe /i <YourProjectName>ConfigDialog.dll
```
(if you have renamed this DLL, you need to reinstall it)

In addition, ensure the following assemblies are still in the GAC:
```
gacutil.exe /l Simba.DotNetDSI
gacutil.exe /l Simba.DotNetDSIExt
```

You will see the message, "Assembly successfully added to the cache" if the operation was successful.

Note: If your driver is already installed in the GAC, then it must be uninstalled from GAC before installing it again. Run the following command (as administrator):

```
gacutil.exe /u <YourProjectName>DSII
```

Update the registry

To update the registry keys, do the following:

1. In Microsoft Visual Studio, click **File** > **Open** > **File** and navigate to
```
[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\<YourProjectName>\
Source.
```

2. For 32-bit Windows, open `SetupMyDotNetQuickstartDSII_32on32.reg`.

   For a 32-bit ODBC driver on 64-bit Windows, open
   `SetupMyDotNetQuickstartDSII_32on64.reg`.

   For a 64-bit ODBC driver on 64-bit Windows, open
   `SetupMyDotNetQuickstartDSII_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, you would 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\...\ODBC Data Sources]` section, change `"MyDotNetQuickstartDSII"="MyDotNetQuickstartDSIIDriver"` 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\...\ODBC.INI\MyDotNetQuickstartDSII]` so that it contains your new data source name. For example, `[HKEY_LOCAL_MACHINE\...\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\...\ODBCINST.INI\ODBC Drivers]` section, change `"MyDotNetQuickstartDSIIDriver"="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\...\ODBCINST.INI\MyDotNetQuickstartDSIIDriver]` so that it contains your new driver name. For example, `[HKEY_LOCAL_MACHINE\...\ODBCINST.INI\<YourProjectName>DSIIDriver]`

9. Beside 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**. Then, replace "`DotNetQuickstart`" in the whole file with the name of your new ODBC driver.

11. Click **Save** and then close the file.

12. In the Registry Editor (`regedit.exe`), click **File** > **Import**, navigate to the registry file that you just modified and then click **Open**.
    A message is displayed that says that the keys and values have been successfully added to the registry.

View your new data source 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 (other than Windows 8), 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 `%WINDIR%\SysWOW64\odbcad32.exe`. See Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 23 for details.

2. In the ODBC Data Source Administrator, click the **System DSN** tab.

3. Scroll through the list of System Data Sources, select **<YourProjectName>DSII** and then 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.

Test your new data source

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 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. 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, verify that the **Attach to** field is set to "Managed code". In the list of available processes, select the ODBC Test process and then click **Attach**. The process name will be either `odbc32.exe` or `odbct32w.exe`.

4. Add a breakpoint on the `QSConnection.cs` constructor. This code 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 then 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.

Day Two

Today's goal is to customize your driver, enable logging and establish a connection to your data store. Several "TODO" comments appear in the source code. They instruct you how to modify the sample driver.

View the list of TODO messages

1. Go to Microsoft Visual Studio and then click **Edit > Find and Replace > Find in Files**.

2. In the Find and Replace window, in the **Find what** text box, type TODO and then click **Find All**. The results are displayed in the Find Results output window.

3. Double-click the entry in the Find Results window to jump to that line in the code.

    The list of TODO messages is as follows:

    ```
    TODO #1: Construct driver singleton. (QuickstartCLIDSI.cpp)
    TODO #2: Set the driver properties.  (QSDriver.cs)
    TODO #3: Check Connection Settings.  (QSConnection.cs)
    TODO #4: Establish A Connection.     (QSConnection.cs)
    TODO #5: Create and return your Metadata Sources.     (QSDataEngine.cs)
    TODO #6: Create and return your Metadata Helper. (QSDataEngine.cs)
    TODO #7: Open A Table.   (QSDataEngine.cs)
    TODO #8: Set the vendor name, which will be prepended to error messages.
      (QSDriver.cs)
    ```

Construct a driver singleton

```
TODO #1: Construct driver singleton.
```

The `LoadDriver()` implementation in `QuickstartCLIDSI.cpp` in the `QuickstartCLIDSI` project is the main hook that is called from Simba's ODBC layer to create an instance of your DSI implementation. Note that the QuickstartCLIDSI library is a C++ CLI library and is therefore able to construct an instance of a managed class. This method is called as soon as the Driver Manager calls `LoadLibrary()` on your ODBC driver.

1. In Microsoft Visual Studio, open the file that contains the TODO #1 message.

2. Look at the `LoadDriver()` implementation and replace "Simba" with your company name and change "DotNetQuickstart" to the name of your driver in the following line:

    ```
    SimbaSettingReader::SetConfigurationBranding("Simba\\DotNetQuickstart");
    ```

`SetConfigurationBranding` changes the registry location that will be used when reading driver settings from the registry. By default, it looks in HKLM\SOFTWARE[\Wow6432Node]\Simba\Driver for a driver, where the Wow6432Node section is used for a 32-bit driver on a 64-bit windows machine. When you change the branding by using a string such as "Company\Driver", then it will look in HKLM\SOFTWARE[\Wow6432Node]\Company\Driver. This is where the ErrorMessagesPath and other required registry settings will be placed.

3. If the driver is running as a server (SERVERTARGET is defined) then you can update the service name from "SimbaDotNetQuickstartService" to the name of your new service.

4. You may want to add processing at this point if you are building a commercial driver.

5. Click **Save**.

Set the driver properties

```
TODO #2: Set the driver properties.
```

1. Double click the TODO #2 message to jump to the relevant section of code.
   The `QSDriver.cs` file opens. Look at `SetDriverPropertyValues()` where you will set up the general properties for your driver. The available driver properties are defined in the DriverPropertyKey enum.

2. Change the DSI_DRIVER_DRIVER_NAME setting. Set this to the name of your driver.

   Note: You may want to revisit this section when fully productizing your driver.

Check the connection settings

```
TODO #3: 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.

The entries from the DSN are only included if a DSN is specified in the connection string instead of a Driver or if the ODBC connection method explicitly uses the DSN.

1. Double click the TODO #3 message to jump to the relevant section of code.

2. The `UpdateConnectionSettings()` function should validate that the key-value pairs in `requestSettings` are sufficient to create a connection. Use the `VerifyRequiredSetting()` or `VerifyOptionalSetting()` utility functions to do this.
   For example, the driver verifies that the entries within `requestSettings` are sufficient to create a connection, by using the following code:

   ```
   VerifyRequiredSetting(DBF_KEY, requestSettings, responseSettings);
   ```

   The example driver requires a single connection key, called "DBF_KEY", which represents the file location to be searched. It is used to look up the DBF path in the connection string.

   If the entries within `requestSettings` are not sufficient to create a connection, then you can ask for additional information from the ODBC-enabled application by specifying, in the `responseSettings` return value, the additional information required.

Establish a connection

```
TODO #4: Establish A Connection.
```

Once `QSConnection::UpdateConnectionSettings()` returns a Dictionary `<string, ConnectionSetting>` object without any **required** settings (if there are only optional settings, a connection can still occur), the Simba ODBC layer will call `QSConnection::Connect()` passing in all the connection settings received from the application.

1. Double click the TODO #4 message to jump to the relevant section of code.

2. Modify the code to authenticate the user against your data store using the information provided within the `requestSettings` parameter. The sample code uses the utility functions `GetRequiredSetting()` and `GetOptionalSetting()` to retrieve the appropriate settings. `GetRequiredSetting`() fetches a required setting from the passed in settings, and will throw an authorization exception if the setting is not present. `GetOptionalSetting()` will fetch an optional setting from the passed in settings and will not throw an exception if the setting is not present. These settings can be used to fetch the needed settings from the passed in ones to create a full connection to the underlying data-source.

    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

Create and return metadata sources

```
TODO #5: Create and return your Metadata Sources.
```

`QSDataEngine::MakeNewMetadataSource()` 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 the following:

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

Handle DSI_TYPE_INFO_METADATA

The underlying ODBC catalog function SQLGetTypeInfo is handled as follows:

1. When called with `DSI_TYPE_INFO_METADATA`, `QSDataEngine::MakeNewMetadataSource()` will return an instance of `QSTypeInfoMetadataSource()`.
2. The example driver 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::InitalizeDataTypes()`. You change the passed in SqlTypeInfo object to modify the parameters of the types that are supported.

Handle the other MetadataSources

> TODO #6: Create and return your Metadata Helper.

The other ODBC catalog functions (including SQLTables (CATALOG_ONLY), SQLTables (TABLE_TYPE_ONLY), SQLTables (SCHEMA_ONLY), SQLTables and SQLColumns) are handled automatically by the metadata helper class, as follows:

1. When called with any other `DSIMetadataTableId`, `QSDataEngine::MakeNewMetadataSource()` 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 do not 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 DotNetQuickstart 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` (which you will do in the next section), 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.

Day Four

Today's goal is to enable data retrieval from within the driver. We will cover the process of opening a table defined within your data store, retrieving the column information for the table, and finally retrieving data.

Open a table

```
TODO #7: Open A Table.
```

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, a new instance of `QSTable` is returned.

    `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 DotNetQuickstart Driver has been implemented for Tabbed Unicode Files. It translates the text from UTF16-LE strings into the SQL Data types defined for each column.

2. Make the following changes to `QSTable` for it to work with your data store:

    -   Return the catalog, schema and table names for your table

        o   `QSTable::QSTable()`: The constructor must be modified to take in the catalog, schema and table names and save them in member variables.

        o   `QSTable::CatalogName (Property)`: Returns `Quickstart.CATALOG`;

        o   `QSTable::SchemaName (Property)`: Returns `null` (because it does not support schemas);

        o   `QSTable::TableName (Property)`: Returns `m_Name`;

    -   Return the columns defined for your table

        o   `QSTable::InitializeColumns()`: This method must be modified so that, for each column defined in the table, you define a `DSIColumn` in terms of SQL types.

    Here is an example of pseudo code for the new method:

```
private IList<IColumn> m_Columns = new List<IColumn>();

Get all column information from your data store for the table

For Each Defined Column
{
  // Change the first parameter of this method to the SQL
  // type that maps to your data store type.
```

```
      TypeMetadata typeMetadata = TypeMetadata.CreateTypeMetadata
          (SqlType sqlType);

        DSIColumn column = new DSIColumn(typeMetadata, name, TableName,
          SchemaName, CatalogName);

        m_Columns.Add(column);
      }
```

- Data Retrieval
  - o QSTable::MoveToBeforeFirstRow()
  - o QSTable::MoveToNextRow()
  - o QSTable::GetData()

  These three methods are responsible for navigating a data structure containing information about one table in your data store, and retrieving data from that table.

  It is best to implement a class that provides a streaming interface for the data in the table within your data store. It should also provide the ability to navigate forward from one table row to the next. The class should be able to navigate across columns within the row and to read the data associated with the current row and column combination.

  In the DotNetQuickstart Driver, `QSTable` uses a `TabbedFileReader`, 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.

  - o 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 resources associated with the data retrieval.

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. You also should be able to 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.

Day Five

Today's goal is to start productizing your driver.

Configure error messages

All the error messages used within your DSI implementation are stored in the resource file called `Resource.resx`.

1. Open the `Resource.resx` file. It is located within the Properties folder in Solution Explorer.

2. Update the error messages. Then save and close the file.

Set the vendor name

```
TODO #8: 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 #8 message to jump to the relevant section of code.

2. Set the vendor name as shown in the commented code.

Create a driver configuration dialog

1. The driver configuration dialog is displayed to the user when they use the ODBC Data Source Administrator to create a new ODBC DSN or configure an existing one. The project contains an example ODBC configuration dialog.

   If you create your own configuration dialog, then the **Setup** key in the driver registry entry under `odbcinst.ini` needs to point to the binary containing the dialog and you need to put the C# assembly into the GAC.

2. 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 (other than on Windows 8). 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 %WINDIR%\SysWOW64\odbcad32.exe. See Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 23 for details.

You are now done with all of the TODO's in the project. You have created your own, custom ODBC driver using SimbaEngine by modifying and customizing the DotNetQuickstart sample driver. Now, you have a read-only driver that connects to your data store.

Appendix 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, which is a good thing, because most applications out there are still 32-bit. Microsoft Excel 2010 is one of the few applications (at the time of this writing) to be available in both 64-bit and 32-bit versions, so it is highly likely that you will encounter 32-bit applications running on 64-bit systems.

It is important to understand that 64-bit applications can only load 64-bit drivers and 32-bit applications can only load 32-bit drivers. In a single running process, all of the code must be either 64-bit or 32-bit.

On a 64-bit Windows system, the ODBC Data Source Administrator that you access through the Control Panel can only be used to configure data sources for 64-bit applications. However, the 32-bit version of the ODBC Data Source Administrator must be used to configure data sources for 32-bit applications.   This is the source of many confusing problems where what appears to be a perfectly configured ODBC DSN does not work because it is loading the wrong kind of driver.

> 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 (you may find it useful to put a shortcut to this on your desktop or Start menu if you access it frequently).

Because of this, it is very important, when using 64-bit Windows, that you configure 32-bit and 64-bit drivers using the correct version of the ODBC Data Source Administrator for each.

Appendix B: Windows Registry 32-Bit vs. 64-Bit

As noted previously, the 32-bit and 64-bit drivers must remain clearly separated because you cannot use a 32-bit driver from a 64-bit application or vice versa. The 32-bit and 64-bit ODBC drivers are installed and data source names are created in different areas of the registry:

32-Bit Drivers on 32-Bit Windows

The Data Source Names and Driver Locations that are relevant to the C# examples for this document are detailed below.

Data Source Names

To connect your driver to your database, the 32-bit ODBC Driver Manager on 32-bit Windows uses Data Source Name registry keys in `HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBC.INI`. The default `[INSTALL_DIRECTORY]` is `C:\Simba Technologies`.

The keys that are relevant to the C# examples discussed in this document are:

- **DotNetQuickstartDSII** which includes the following key names and values:
  - **Driver:**
    `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\Win32\`
    `Release_MTDLL\QuickstartCLIDSI_MTDLL.dll`
  - **DBF:**
    `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Databases\Quickstart`
  - **Description:** `Sample 32-bit SimbaEngine DotNetQuickstart DSII`
- **DotNetUltraLightDSII** which includes the following key names and values:
  - **Driver:**
    `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\Win32\`
    `Release_MTDLL\UltraLightCLIDSI_MTDLL.dll`
  - **Description:** `Sample 32-bit SimbaEngine DotNetUltraLight DSII`

There is another registry key at the same location called ODBC Data Sources. String values that correspond to each DSN/driver pair must also be added to it:

- **ODBC Data Sources** which includes the following key names and values:
  - **DotNetQuickstartDSII:** `DotNetQuickstartDSIIDriver`
  - **DotNetUltraLightDSII:** `DotNetUltraLightDSIIDriver`

Driver Locations

To define each driver and its setup location, the 32-bit ODBC Driver Manager on 32-bit Windows uses registry keys created in `HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI`. Each key includes three string values to define the location of the **Driver**, its **Setup** location and the

Description to help you clearly identify each registry key. The keys that are relevant to the C# examples discussed in this document are:

- **DotNetQuickstartDSIIDriver** which includes the following key names and values:

    o **Driver:**
      `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\Win32\Release_MTDLL\QuickstartCLIDSI_MTDLL.dll`

    o **Setup:**
      `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\Win32\Release_MTDLL\QuickstartCLIDSI_MTDLL.dll`

    o **Description:** `Sample 32-bit SimbaEngine DotNetQuickstart DSII`

- **DotNetUltraLightDSIIDriver** which includes the following key names and values:

    o **Driver:**
      `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\Win32\Release_MTDLL\UltraLightCLIDSI_MTDLL.dll`

    o **Description:** `Sample 32-bit SimbaEngine DotNetUltraLight DSII`

There is another registry key at the same location called ODBC Drivers, indicating which drivers are installed. String values that correspond to each driver must also be added to it:

- **ODBC Drivers** which includes the following key names and values:

    o **DotNetQuickstartDSIIDriver:** `Installed`
    o **DotNetUltraLightDSIIDriver:** `Installed`

32-Bit Drivers on 64-Bit Windows

The 32-bit applications and drivers use a section of the registry that is separate from the 64-bit applications and drivers. Note that from the point of view of a 32-bit application on a 64-bit machine, 32-bit data sources look exactly like they do on a 32-bit machine.

Data Source Names

To connect your driver to your database, the 32-bit ODBC Driver Manager on 64-bit Windows uses Data Source Name registry keys in
`HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBC.INI`. The keys that are relevant to the C# examples discussed in this document are:

- **DotNetQuickstartDSII** which includes the following key names and values:

    o **Driver:**
      `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\Win32\Release_MTDLL\QuickstartCLIDSI_MTDLL.dll`

    o **DBF:**
      `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Databases\Quickstart`

    o **Description:** `Sample 32-bit SimbaEngine DotNetQuickstart DSII`

- **DotNetUltraLightDSII** which includes the following key names and values:

  o **Driver**:
   `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\Win32\`
   `Release_MTDLL\UltraLightCLIDSI_MTDLL.dll`

  o **Description:** `Sample 32-bit SimbaEngine DotNetUltraLight DSII`

There is another registry key at the same location called ODBC Data Sources. String values that correspond to each DSN/driver pair must also be added to it:

- **ODBC Data Sources** which includes the following key names and values:

  o **DotNetQuickstartDSII:** `DotNetQuickstartDSIIDriver`
  o **DotNetUltraLightDSII:** `DotNetUltraLightDSIIDriver`

Driver Locations

To define each driver and its setup location, the 32-bit ODBC Driver Manager on 64-bit Windows uses registry keys created in
`HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBCINST.INI.` Each key includes three string values to define the location of the **Driver**, its **Setup** location and the **Description** to help you clearly identify each registry key. The keys that are relevant to the C# examples discussed in this document are:

- **DotNetQuickstartDSIIDriver** which includes the following key names and values:

  o **Driver**:
   `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\win32\`
   `Release_MTDLL\QuickstartCLIDSI_MTDLL.dll`

  o **Setup:**
   `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\win32\`
   `Release_MTDLL\QuickstartCLIDSI_MTDLL.dll`

  o **Description:** `Sample 32-bit SimbaEngine DotNetQuickstart DSII`

- **DotNetUltraLightDSIIDriver** which includes the following key names and values:

  o **Driver:**
   `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\Win32\`
   `Release_MTDLL\UltraLightCLIDSI_MTDLL.dll`

  o **Description:** `Sample 32-bit SimbaEngine DotNetUltraLight DSII`

There is another registry key at the same location called ODBC Drivers, indicating which drivers are installed. String values that correspond to each driver must also be added to it:

- **ODBC Drivers** which includes the following key names and values:

  o **DotNetQuickstartDSIIDriver:** `Installed`
  o **DotNetUltraLightDSIIDriver:** `Installed`

64-Bit Drivers on 64-Bit Windows

The Data Source Names and Driver Locations that are relevant to the C# examples for this document are detailed below.

Data Source Names

To connect your driver to your database, the 64-bit ODBC Driver Manager on 64-bit Windows uses Data Source Name registry keys in `HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBC.INI`. The keys that are relevant to the examples discussed in this document are:

- **DotNetQuickstartDSII** which includes the following key names and values:

  o **Driver:** `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\x64\Release_MTDLL\QuickstartCLIDSI_Release_MTDLL.dll`

  o **DBF:** `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Databases\Quickstart`

  o **Description:** `Sample 64-bit SimbaEngine DotNetQuickstart DSII`

- **DotNetUltraLightDSII** which includes the following key names and values:

  o **Driver:** `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\x64\Release_MTDLL\UltraLightCLIDSI_Release_MTDLL.dll`

  o **Description:** `Sample 64-bit SimbaEngine DotNetUltraLight DSII`

There is another registry key at the same location called ODBC Data Sources. String values that correspond to each DSN/driver pair must also be added to it:

- **ODBC Data Sources** which includes the following key names and values:

  o **DotNetQuickstartDSII:** `DotNetQuickstartDSIIDriver`
  o **DotNetUltraLightDSII:** `DotNetUltraLightDSIIDriver`

Driver Locations

To define each driver and its setup location, the 64-bit ODBC Driver Manager on 64-bit Windows uses registry keys created in `HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI`. Each key includes three string values to define the location of the **Driver**, its **Setup** location and the **Description** to help you clearly identify each registry key. The keys that are relevant to the C# examples discussed in this document are:

- **DotNetQuickstartDSIIDriver** which includes the following key names and values:

  o **Driver:** `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\x64\Release_MTDLL\QuickstartCLIDSI_Release_MTDLL.dll`

  o **Setup:** `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\x64\Release_MTDLL\QuickstartCLIDSI_Release_MTDLL.dll`

  o **Description:** `Sample 64-bit SimbaEngine DotNetQuickstart DSII`

- **DotNetUltraLightDSIIDriver** which includes the following key names and values:

  o **Driver:** `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Builds\Bin\x64\Release_MTDLL\UltraLightCLIDSI_Release_MTDLL.dll`

  o **Description:** `Sample 64-bit SimbaEngine DotNetUltraLight DSII`

There is another registry key at the same location called ODBC Drivers, indicating which drivers are installed. String values that correspond to each driver must also be added to it:

- **ODBC Drivers** which includes the following key names and values:
    - **DotNetQuickstartDSIIDriver:** `Installed`
    - **DotNetUltraLightDSIIDriver:** `Installed`

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:

1. Each `MetadataSource` implementation of `GetMetadata()`

2. `QSTable::GetData()`

These methods accept the following parameters:

- **column**: Uniquely identifies a column within the current row.   For MetadataSource, the Simba SQL Engine will pass in a unique column tag (see DSIOutputMetadataColumnTag). For QSTable, the Simba SQL Engine will pass in the column index. The first column uses index 0.

- **offset**: The number of bytes in the data to skip before copying data into the out_data parameter. 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.

- **maxSize**: The maximum number of bytes of data to copy into the out_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.

- **out_data**: The data to be returned.

  Note: "offset" and "maxSize" are only applicable to data that can be retrieved in multiple parts (for example, character or binary) and can be ignored otherwise.

Appendix D: How to Add Schema Support

Microsoft Excel does not require schema support to work properly with your new driver. However, some other applications require schema support and if your data store supports schemas then you might want to provide access to them for your users. The following instructions describe how to add schema support to your new ODBC driver.

Handling DSI_SCHEMAONLY_METADATA

1. By default, `QSConnnection::SetConnectionProperties()` disables schema support via `PropertyUtilities::SetSchemaSupport()`. Change this value to `true` to enable schema support.

2. You will also need to change:

   a. `QSMetadataHelper::GetNextTable()`
      In the SimbaEngine DotNetQuickstart driver, a blank schema is returned as schema support is not enabled by default. The schema will need to be returned in the Identifier to allow SimbaEngine to open the correct table.

   b. `QSDataEngine::OpenTable()`
      Modify this method to verify the given schema and return the correct table for the given catalog, schema, and table name.

   c. `QSTable::SchemaName()`
      Modify the get method of this property to return the schema that the table belongs to.

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