Magnitude Simba SDK

Build a C# ODBC Driver for SQL-Capable Data Sources in 5 Days (Windows)

Version 10.2.2

October 2022

Copyright

Contact Us

Magnitude Software, Inc.

www.magnitude.com

About this Guide

Purpose

This guide explains how to use the Magnitude Simba SDK to create a custom ODBC connector for a data store that is SQL-aware. It explains how to customize the UltraLight sample connector, which is included with the Simba SDK.

Using this sample connector is the quickest and easiest way to create a custom ODBC connector. At the end of five days, you will have a read-only connector that connects to your data store. This custom ODBC connector can be used as the foundation for a commercial DSI implementation.

Note:

An online version of this guide is located at http://www.simba.com/resources/sdk/documentation.

Advantages of Using the Simba SDK

The ODBC specification defines a rich interface that allows any ODBC-enabled application to connect to a data store. In order to implement a connector that supports this specification, developers have to understand all the complexities of error checking, session management, and data conversion, then design their code in a robust and efficient manner. Developers must also understand how to optimize data retrieval in order to get maximum performance when connecting to large and complex data stores.

The Simba SDK, developed by experts in the field, is a complete implementation of the ODBC specification. It exposes an easy-to-use SDK that allows you to create a robust and efficient connector for your data store.

Build a Custom ODBC Connector in Five Days

Over the course of five days, this guide explains how to accomplish the following tasks:

1. Set up the development environment and build the sample connector.
2. Use the sample connector as a template to create a custom ODBC connector.
3. Make a connection to the data store.
4. Retrieve metadata.
5. Work with columns.

6. Retrieve data.
7. Rename and rebrand the custom ODBC connector.

In the UltraLight connector, the areas of code that require modification are marked with “TODO” messages and a short explanation. Some of these changes customize the connector for your specific data store, while other changes rename the connector for your company or product.

Audience

The guide is intended for developers who want to use the Simba SDK to build a connector for a data store that is SQL-aware.

Document Conventions

Italics are used when referring to book and document titles.

Bold is used in procedures for graphical user interface elements that a user clicks and text that a user types.

`Monospace font` indicates commands, source code or contents of text files.

> **NOTE:**
>
> Indicates a short note appended to a paragraph.

> **IMPORTANT:**
>
> Indicates an important comment related to the preceding paragraph.

Knowledge Prerequisites

To use the Simba SDK to build a custom ODBC connector, the following knowledge is helpful:

- Familiarity with the C# programming language.
- Ability to use the data store to which the connector you are developing will connect.
- An understanding of the role of ODBC technologies and driver managers in connecting to a data store.
- Exposure to SQL.

Variables Used in this Document

The following variables are used in this document:

Variable	Description
[INSTALL_DIR]	Installation directory for the SimbaEngine X SDK. **Default value on Windows platforms**: `C:\Simba Technologies\SimbaEngineSDK\10.2` **Default value on Linux, Unix, and macOS platforms**: *[UNTAR_DIR]*`/SimbaEngineSDK/10.2`
[PROJ_NAME]	The name of your own sample project (which is also the name of your project directory).
[DRIVER_NAME]	The name of your own driver.
[UNTAR_DIR]	Directory where the SimbaEngine X SDK distributable was untarred.

Introduction

This guide will show you how to create your own, custom ODBC driver using the SimbaEngine SDK. 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 the SimbaEngine X SDK

The SimbaEngine X SDK 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 the SimbaEngine X SDK 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 the SimbaEngine X SDK is available on the Simba website at http://www.simba.com/odbc-sdk-documents.htm.

You use the SimbaEngine X SDK 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 X SDK 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 X SDK 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 UltraLight sample driver

The UltraLight 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-capable data source, the Simba SQLEngine component must be used to perform the necessary SQL processing.

The UltraLight driver helps you to prototype a DSI implementation for your own data store so you can learn how the SimbaEngine X SDK 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.

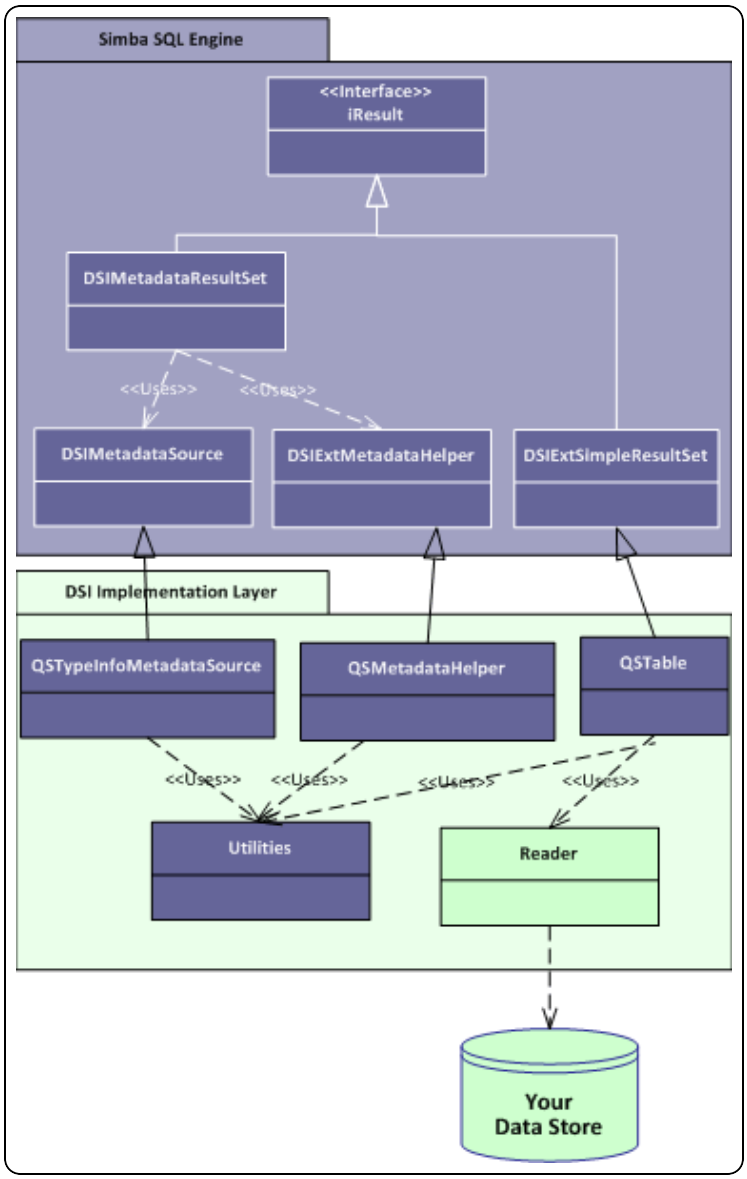


Figure 1: Design pattern for a DSI implementation.

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.

Overview

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

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 UltraLightdriver, 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 driver customization, rather than connecting to the Simba SQL engine. These are tasks such as 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 UltraLightdriver 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 the SimbaEngine X SDK

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

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

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

Build the UltraLight 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 `[INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\DotNet ultralight\Source` and then open the `ultralight_Driver_vs2013.sln` file.
   The default `[INSTALLDIR]` is `C:\Simba Technologies`.

   The solution contains two projects: **UltraLight_Driver_vs2013**, which is the C# driver implementation, and **UltraLightCLIDSI**, which is the driver's native component (C++ CLI) and is the ODBC driver. Every .NET ODBC driver built using the SimbaEngine X SDK will have these two components, one managed and one native. The native component (in this case UltraLightCLIDSI) has only

one function, to create an instance of the .NET driver object from the implementation.

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**.

> **NOTE:** Visual Studio 2013 has different command prompts for different targets which are named accordingly (e.g. VS2013 x64 Cross Tools Command Prompt). Choose the command prompt which corresponds to your target.

3. Change to the directory that contains the DLL file. Enter a command using the following examples:
   - For 32-bit drivers: `cd [INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\DotNet ultralight\Bin\Win32\Debug_MTDLL`
   - Or for 64-bit drivers: `cd [INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\DotNet ultralight\Bin\x64\Debug_MTDLL`
4. Type the following command to install the assembly into the GAC:
   ```
   gacutil.exe /i Simba.UltraLight.Driver.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, Simba.DotNetDSI.dll and Simba.DotNetDSIExt.dll have to be installed in the GAC. These files were installed in the GAC during SDK 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 X SDK installer

The SimbaEngine X SDK 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
- **UltraLightDSII** - 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
- **UltraLightDSIIDriver** - 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`
   - Or 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`
   - Or 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 Windows Registry 32-Bit vs. 64-Bit on page 33.

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, click **Start > Control Panel > 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 (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**. See32-Bit Drivers on 64-Bit Windows on page 34.
2. In the ODBC Data Source Administrator, click the **System DSN** tab.
3. Scroll through the list of System Data Sources, select **DotNetUltraLightDSII** 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 the SDK, 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\`

2. Navigate to the folder that corresponds to your driver's architecture (amd64, ia64 or x86).
3. Choose one of the following:
   - To launch the ANSI version click **odbcte32.exe**, or
   - Or to launch the Unicode version, click **odbct32w.exe**.

   NOTE: It is important to run the correct version of the ODBC Test tool for ANSI or Unicode and 32-bit or 64-bit.
4. In the ODBC Test tool, select **Conn > Full Connect**.
   The Full Connect window opens.
5. Select the **UltraLightDSII 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.
6. When the tool connects to the data source, you will see the message, **Successfully connected to DSN 'DotNetultralightDSII'**.

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 the SDK, 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.

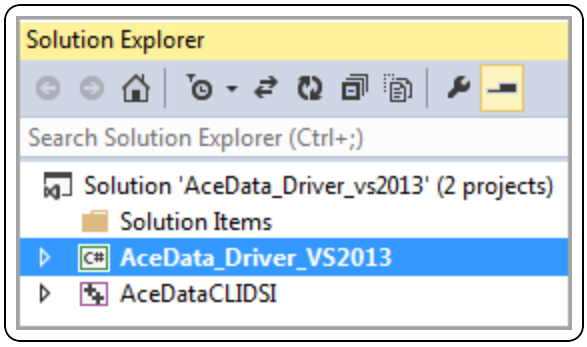
To rename the project:

1. In your Windows Explorer window, copy the `[INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\DotNet ultralight` directory and paste it to the same location. This will create a new directory called DotNetUltraLight - 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 **ultralight_Driver_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 the **UltraLight_Driver_vs2013** solution to ***<YourProjectName>*_Driver_VS2013**.
   - Rename the **C# project** ultralight_**Driver_vs2013** to ***<YourProjectName>*_Driver_VS2013**.
   - Rename ultralight**CLIDSI** to ***<YourProjectName>*CLIDSI**.

   For example, if your project name was AceData, your solution might look like this:



6. Right click on ***<YourProjectName>*_Driver_VS2013** and select properties.
7. In the Assembly name text box, replace **Simba.UltraLight.Driver** with ***<YourProjectName>*Driver**.
8. Click **File > Save All**.

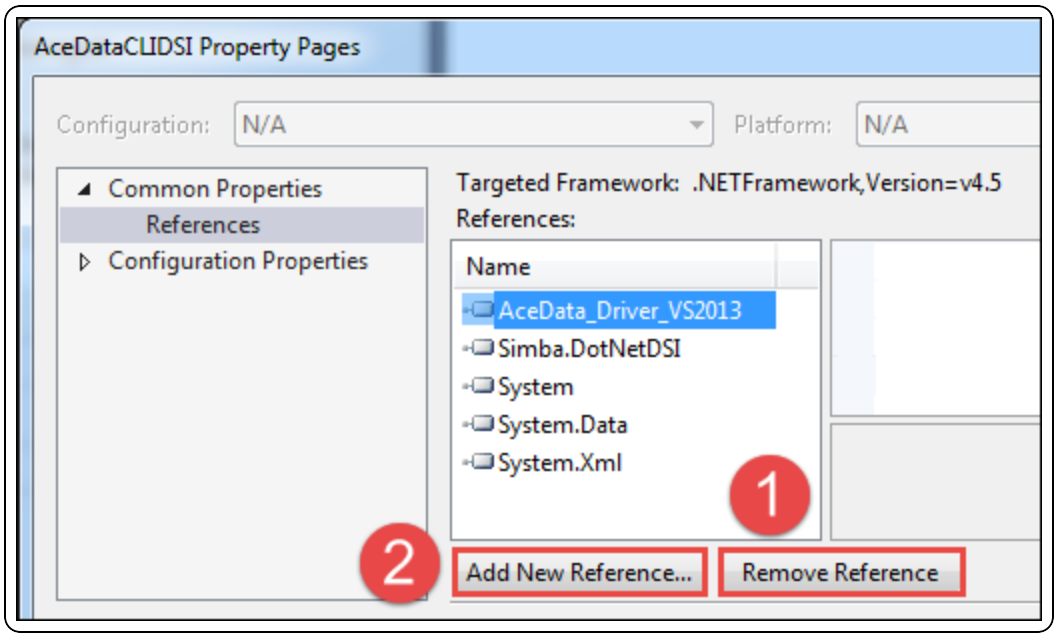
Your project is renamed, but you still need to update the namespaces.

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. 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. Click **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**.

> **NOTE:** Visual Studio 2013 has different command prompts for different targets which are named accordingly (e.g. VS2013 x64 Cross Tools Command Prompt). Choose the command prompt which corresponds to your target.

3. Change to the directory that contains the DLL file. Type a command using one of the following examples:
   - For 32-bit drivers: `cd [INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\<YourProjectName>\Bin\Win32\Debug_MTDLL`
   - Or for 64-bit drivers: `cd [INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\<YourProjectName>\Bin\x64\Debug_MTDLL`
4. Type the following commands to install the assemblies into the GAC:
   ```
   gacutil.exe /i <YourProjectName>.Driver.dll
   gacutil.exe /i <YourProjectName>ConfigDialog.dll
   ```

   > **NOTE:** If you have renamed this DLL, you need to reinstall it.

   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 `[INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\<YourProjectName>\Source`.
2. Choose one of the following:
   - For 32-bit Windows, open **SetupMyDotNetUltraLightDSII_32on32.reg**.
   - For a 32-bit ODBC driver on 64-bit Windows, open **SetupMyDotNetUltraLightDSII_32on64.reg**.

- For a 64-bit ODBC driver on 64-bit Windows, open **SetupMyDotNetUltraLightDSII_64on64.reg**.

3. In the file, replace [INSTALLDIR] 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 [INSTALLDIR] 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 **"MyDotNetultralightDSII"="MyDotNetultralightDSIIDriver"** 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\MyDotNetultralighttDSII]** 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 **"MyDotNetultralightDSIIDriver"="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\MyDotNetultralightDSIIDriver]** so that it contains your new driver name. For example, **[HKEY_LOCAL_MACHINE\...\ODBCINST.INI\<YourProjectName>DSIIDriver]**
9. Beside the line that starts with **Driver**, update the path to the DLL file.

   Note:

   This is not the same DLL file that was added to the Global Assembly Cache (GAC).

10. Click **Edit > Find and Replace > Quick Replace**. Then, replace **DotNetultralight** 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, click **Control Panel > 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 (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 32-Bit Drivers on 32-Bit Windows on page 33 .
2. In the ODBC Data Source Administrator, click the **System DSN** tab.
3. Scroll through the list of System Data Sources, select ***<YourProjectName>*DSII**.
4. Now that you have located 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\`
2. 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.
3. Attach **Visual Studio** to the **ODBC Test** process. To do this, go to **Microsoft Visual Studio** and then click **Debug > Attach to Process**
4. In the Attach to Process window, verify that the **Attach** to field is set to both Managed (v4.5, v4.0) Code and Native 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**.
5. Add a breakpoint on the **ULConnection.cs** constructor. This code runs as soon as the **Driver Manager** loads the ODBC driver.
6. In the **ODBC Test** tool, select **Conn > Full Connect**.
   The **Full Connect** window opens.
7. 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.
8. You should hit the breakpoint you created and focus should switch to **Visual Studio**.
9. 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 (ultralightCLIDSI.cpp).
   - TODO #2: Set the driver properties (ULDriver.cs).
   - TODO #3: Check connection settings (ULConnection.cs).
   - TODO #4: Establish a connection (ULConnection.cs).
   - TODO #5: Create and return your Metadata Sources (ULDataEngine.cs).
   - TODO #6: Prepare a query (ULDataEngine.cs).
   - TODO #7: Implement a QueryExecutor (ULQueryExecutor.cs).
   - TODO #8: Provide parameter information (ULQueryExecutor.cs).
   - TODO #9: Implement Query Execution (ULQueryExecutor.cs).
   - TODO #10: Implement your DSISimpleResultSet (ULPersonTable.cs).
   - TODO #11: Set the vendor name, which will be prepended to error messages (ULDriver.cs).

Construct a driver singleton

TODO #1: Construct driver singleton

The `LoadDriver()` implementation in `ultralightCLIDSI.cpp` in the `ultralightCLIDSI` project is the main hook that is called from Simba's ODBC layer to create an instance of your DSI implementation. Note that the `UltraLightCLIDSI` 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 UltraLightto the name of your driver in the following line:

   ```
   SimbaSettingReader.SetConfigurationBranding
   ("Simba\\DotNetultralight");
   ```

   `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. The `\Driver` or `\Server` suffix is added depending on configuration.
3. If the driver is running as a server (`SERVERTARGET` is defined) then you can update the service name from `SimbaDotNetultralightService` 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 `ULDriver.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

`ULConnection.UpdateConnectionSettings()` method which is responsible for verifying that all of the required, and any optional, connection settings are present.

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.

For example, the connection string `DSN=UltraLight;UID=user` will be broken down into key value pairs and passed in via the connectionSettings parameter. In this case that dictionary would contain two entries: `{DSN, UltraLight}` and `{UID, user}`. If a DSN was specified, then the DSN value is removed from the map and any entries that are stored in the preconfigured DSN are inserted into the map. Once the map has been created with all the key-value pairs from the connection string and DSN, this map is passed down to the DSII.

1. Double click the **TODO #3** message to jump to the relevant section of code.
2. The `UpdateConnectionSettings()` method should validate that the key-value pairs in `requestSettings` are sufficient to create a connection. Use the `VerifyRequiredSetting()` or `VerifyOptionalSetting()` utility methods 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(UltraLight.UL_UID_KEY,
   requestSettings, responseSettings);
   VerifyRequiredSetting(UltraLight.UL_PWD_KEY,
   requestSettings, responseSettings);
   VerifyOptionalSetting(UltraLight.UL_LNG_KEY,
   requestSettings, responseSettings);
   ```

The example driver requires connection keys for the user name and password to use for connecting, while the language can optionally be specified.

Note that settings can alternatively be verified manually. 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 manually specifying, in the `responseSettings` return value, the additional information required.

Use the DriverPrompt Dialog to get settings

Depending on how the connection was initiated by the application, the SDK may call `ULConnection.PromptDialog()` to allow the user to specify more information. In general, if there are any required settings present in `responseSettings`, then `PromptDialog()` will be called. Note that, if the application requests, `PromptDialog()` may not be called in this case or may be called even if there are no settings in `responseSettings`.

`ULConnection.PromptDialog()` displays a configuration dialog box which is displayed by the **Windows ODBC Data Source Administrator** when configuring the driver.

The method takes in the following:

- **connResponseMap**: a connection response map which can be populated with settings which haven't been entered by the user. This is then used by the driver to notify the user that information is missing. Currently this variable is unused in the sample.
- **connectionSettings**: a connection settings map which is populated by the dialog with settings entered by the user.
- **parentWindow**: the handle to the parent Window to make the prompt window a child of.
- **promptType**: an enum specifying if only required fields are to be available, or if optional fields should be available as well.

The dialog and the related code in this method can be modified to take in different parameters as required by your driver.

Establish a connection

TODO #4: Establish A Connection

Once `ULConnection.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 `ULConnection.Connect()` passing in all the connection settings received from the application.

During `Connect()`, you should have all the settings necessary to make a connection as verified by `UpdateConnectionSettings()`. You can use the utility functions `GetRequiredSetting()` and `GetOptionalSetting()` to request the required and optional settings for your connection, and attempt to make an actual connection.

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 method `GetRequiredSetting()` 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. Another method that can be used is `GetOptionalSetting()` will fetch an optional setting from the passed in settings and will not throw an

exception if the setting is not present. These can be used to fetch the settings passed in 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 at least the following ODBC catalog functions:

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

These catalog functions are represented in the DSI by metadata sources, one for each of the catalog functions.

Create and return metadata sources

TODO #5: Create and return your Metadata Sources

`ULDataEngine.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 MetadataSourceID, which is then mapped to an underlying IMetadataSource that you will implement and return. Each IMetadataSource 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: `MoveToNextRow()`
- Retrieving data: `GetMetadata()` (See the section, Data Retrieval on page 37, for a brief overview of data retrieval). Each column in the metadata source will be represented by a `MetadataSourceColumnTag` which is passed into `GetMetadata()`.

Handle DSI_TYPE_INFO_METADATA

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

1. When called with `DSI_TYPE_INFO_METADATA`, `ULDataEngine.MakeNewMetadataSource()` will return an instance of `ULTypeInfoMetadataSource()`.

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_BIT	SQL_CHAR	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 `ULTypeInfoMetadataSource.InitializeDataTypes()`.

Handle the other MetadataSources

The other ODBC catalog functions (including `SQLTables (CATALOG_ONLY)`, `SQLTables (TABLE_TYPE_ONLY)`, `SQLTables (SCHEMA_ONLY)`, `SQLTables and SQLColumns)` are handled as follows:

1. When called with the corresponding metatable ID's, `ULDataEngine.MakeNewMetadataSource()` returns a new instance of one of the following respective DSIMetadataSource-derived classes:
    - `ULCatalogOnlyMetadataSource`: returns a list of all catalogs. The sample implementation returns one row of information with one column containing the name of a fake catalog. This demonstrates how to return a catalog name.
    - `DSITableTypeOnlyMetadataSource`: (default implementation by Simba) returns metadata about all tables of a particular type (`TABLE`, `SYSTEM TABLE`, and `VIEW`) in the datasource. This class provides two constructors which allow for returning the default set of table types (listed above) or for specifying your own set of table types.
    - `ULSchemaOnlyMetadataSource`: returns a list of all schemas. The sample implementation returns one row of information with one column containing the name of a fake schema. This demonstrates how to return a schema name.
    - `ULTablesMetadataSource`: returns metadata about all of the tables in the data source. The sample hard codes and returns information for the hard coded person table to demonstrate how to return table metadata.

- `ULColumnsMetadataSource`: returns metadata for the columns in the data source. The sample hard codes and returns information for the three columns in the person table consisting of the name column, an integer column, and a numeric column.

2. When called with any other MetadataSourceID, which doesn't correspond to these tables, `ULDataEngine.MakeNewMetadataSource()` returns a new instance of `DSIEmptyMetadataSource` to indicate that no metadata is available for the specified table ID.

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 preparing a query, providing parameter information, implementing a query executor, and implementing a result set.

TODO #6: Prepare a query (ULDataEngine.cs)

The `ULDataEngine.Prepare()` method takes in a query and is expected to pass it to the underlying SQL enabled datasource for preparation. Once prepared, the method then returns a `ULQueryExecutor` which is used by the engine to return results.

For demonstration purposes, the default implementation of `ULDataEngine.Prepare()` performs a very simple preparation by searching for the substrings `select` and `?` in the query. If `select` is found, then it is assumed that the caller wants to search for rows of data and a result set is therefore returned. If `select` is not found, then it is assumed that the caller wants to retrieve the number of rows and so a row count is therefore returned. If `?` is present, then the statement is assumed to be parameterized and therefore `ULQueryExecutor`'s constructor will populate parameters as described below.

The method also checks the query for the string "`{call`" and if found, throws an exception. This demonstrates how to throw an error when parsing a query.

In your implementation you would replace this with more sophisticated logic or pass the query to the data source for preparation.

TODO #7: Implement a QueryExecutor (ULQueryExecutor.cs)

The `ULQueryExecutor` object returned by the `ULDataEngine.Prepare()` method is an implementation of `IQueryExecutor` which, as the name suggests, executes a query. The implementation of `ULQueryExecutor` simply checks if the query passed in contains a select statement or not by looking at the `isSelect` parameter. If `isSelect` is set then the constructor creates and adds a simple result set consisting of people's names to `Results`. Otherwise, it creates and adds a row count.

Modify the implementation to query the data source and store the results.

TODO #8: Provide parameter information (ULQueryExecutor.cs)

`ULExecutorUtilities.CreateParameters()` is called by the `ULQueryExecutor` constructor and handles any parameter information specified when the application calls SQLPrepare. The default implementation shows how to register input, input/output, and output-only parameters. Modify this method as required to register parameters appropriate for your queries.

Note that this method's logic will only be executed if the query contains a parameter and if the hosting application doesn't set `SQL_ATTR_ENABLE_AUTO_IPD` to false.

TODO #9: Implement Query Execution (ULQueryExecutor.cs)

The next step is to handle statement execution in `ULQueryExecutor.Execute()`. The sample delegates this to `ULExecutorUtilities.ExecuteParameters()` which iterates through the input and copies it to the output for consumption by the calling application.

In your implementation, the `Execute()` method should begin by serializing parameters (stored in the Inputs field of the contexts parameter) into a form that the data source can consume. Once this has been done then the data source should then be instructed to execute the statement, after which the results should be placed into the Outputs field of the contexts parameter.

After this method exits, the calling framework will then obtain the results by accessing the `ULQueryExecutor.Results` property.

TODO #10: Implement your DSISimpleResultSet

The final step in returning data is to implement a `DSISimpleResultSet`. The sample contains an implementation called `ULPersonTable` which returns a hardcoded set of people's names.

A `DSISimpleResultSet` implementation contains the data result from a query execution, which the calling framework will use to access each row and column of data.

The implementation should maintain a handle to a cursor within the SQL-enabled data source and delegate calls to the data source to move to the next row when the `MoveToNextRow()` method is called.

In the example, `ULPersonTable.MoveToNextRow()` simply returns whether or not the driver is on the last row of data, so this should be replaced in your implementation with code that delegates this to the data source.

The `GetData()` method is where column data is retrieved, so this should also be modified to extract data from the data source.

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 #11: 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 #11** message to jump to the relevant section of code.
2. Set the vendor name as shown in the commented code.

Finishing Touches

Create a driver configuration dialog

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.

To see the driver configuration dialog that you created, run the ODBC Data Source Administrator. **Control Panel > Administrative Tools > 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 ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 32 for details.

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

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. Microsoft Excel 2010 is one of the few applications 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.

To create new or modify existing 32-bit data sources 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.

Windows Registry 32-Bit vs. 64-Bit

As noted previously, the 32-bit and 64-bit drivers must remain clearly separated because they must match the architecture of the applications that are using them. 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 *`[INSTALLDIR]`* is `C:\Simba Technologies`.

Each key includes string values to define the name of the Driver, a Description to help you clearly identify each registry key, and a Locale to specify the language. The keys that are relevant to the C# examples discussed in this document are:

- `DotNetUltraLightDSII` which includes the following key names and values:
  - Driver: `DotNetUltraLightDSIIDriver`
  - Description: Sample 32-bit SimbaEngine `DotNetUltraLight DSII`
  - Locale: en-US

  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 name and value:
  - `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 string values to define the location of the Driver and a Description to help you clearly identify the registry key. The keys that are relevant to the C# examples discussed in this document are:

- `DotNetUltraLightDSIIDriver` which includes the following key names and values:
- Driver: *[INSTALLDIR]* `\SimbaEngineSDK\10.0\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 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 name and value:

  `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`. Each key includes string values to define the name of the Driver, a Description to help you clearly identify the registry key, and a Locale to specify the language. The keys that are relevant to the C# examples discussed in this document are:

- `DotNetUltraLightDSII` which includes the following key names and values:
- Driver: `DotNetUltraLightDSIIDriver`
- Description: Sample 32-bit SimbaEngine `DotNetUltraLight DSII`
- Locale: `en-US`

  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 name and value:
- `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, and a Description to help you clearly identify the registry key. The keys that are relevant to the C# examples discussed in this document are:

- `DotNetUltraLightDSIIDriver` which includes the following key names and values:
- Driver: *[INSTALLDIR]*`\SimbaEngineSDK\10.0\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 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 name and value:
- `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`. Each key includes string values to define the name of the Driver, a Description to help you clearly identify each registry key, and a Locale to specify the language. The keys that are relevant to the examples discussed in this document are:

- `DotNetUltraLightDSII` which includes the following key names and values:
  - Driver: `DotNetUltraLightlDSIIDriver`
  - Description: Sample 64-bit SimbaEngine `DotNetUltraLight DSII`
  - Locale: `en-US`

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

- `DotNetUltraLightDSIIDriver` which includes the following key names and values:
  - Driver: *[INSTALLDIR]* `\SimbaEngineSDK\10.0\Examples\Builds\Bin\x64\Release_MTDLL\UltraLightCLIDSI_MTDLL.dll`
  - >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 name and value:
  - `DotNetUltraLightDSIIDriver`: Installed

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. `DSISimpleResultSet::GetData()`

These methods accept the following parameters:

- **column**: Uniquely identifies a column within the current row. For MetadataSource, the calling framework will pass in a unique column tag (see MetadataSourceColumnTag). For ULPersonTable, the calling framework 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.

Contact Us

For more information or help using this product, please contact our Technical Support staff. We welcome your questions, comments, and feature requests.

Note:

To help us assist you, prior to contacting Technical Support please prepare a detailed summary of the Simba SDK version and development platform that you are using.

You can contact Technical Support via the Magnitude Support Community at www.magnitude.com.

You can also follow us on Twitter @SimbaTech and @Mag_SW.

Third-Party Licenses

The licenses for the third-party libraries that are included in this product are listed below.

OpenSSL

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:

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.

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/)"

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.

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.

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).

Expat

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.

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.

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

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.

Third-Party Trademarks

Simba, the Simba logo, Simba SDK, 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 and macOS 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.