



# SimbaEngine

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

Version 10.3

August 2024

# Copyright

This document was released in August 2024.

Copyright ©2014–2024 insightsoftware. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from insightsoftware.

The information in this document is subject to change without notice. insightsoftware strives to keep this information accurate but does not warrant that this document is error-free.

Any insightsoftware product described herein is licensed exclusively subject to the conditions set forth in your insightsoftware license agreement.

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

All other company and product names mentioned herein are used for identification purposes only and may be trademarks or registered trademarks of their respective owners.

Information about the third-party products is contained in a third-party-licenses.txt file that is packaged with the software.

## Contact Us

insightsoftware

[www.insightsoftware.com](http://www.insightsoftware.com)

# Table of Contents

Copyright .....	2
Table of Contents .....	3
Introduction .....	5
About SimbaEngine .....	5
About the UltraLight sample driver .....	5
Overview .....	6
<b>Day One – Windows Instructions .....</b>	<b>8</b>
Install SimbaEngine .....	8
Build the UltraLight example driver .....	8
Examine the registry keys added by the SimbaEngine installer .....	9
View the data source in the ODBC Data Source Administrator .....	10
Test the data source .....	10
Set up a new project to build your own ODBC driver .....	11
Build your new driver .....	11
Update the registry .....	12
View your new data source in the ODBC Data Source Administrator .....	14
Test your new data source .....	14
Summary – Day One .....	15
<b>Day Two .....</b>	<b>16</b>
Construct a Connector Singleton .....	16
Set the Connector properties .....	17
Set the logging details .....	17

Check the connection settings .....	17
Customize the DriverPrompt Dialog .....	18
Establish a connection .....	19
<b>Day Three .....</b>	<b>21</b>
Create and return metadata sources .....	21
<b>Day Four .....</b>	<b>24</b>
Prepare and execute a query .....	24
<b>Day Five .....</b>	<b>27</b>
Configure error messages .....	27
Finishing touches .....	27
<b>Reference .....</b>	<b>29</b>
Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit .....	29
Appendix B: Windows Registry 32-Bit vs. 64-Bit .....	29
Appendix C: Data Retrieval .....	31
More information on Date, Time and DateTime types .....	34
Simple Fixed-Length Data Example .....	34
Simple Variable-Length Data Example .....	35
Appendix D: C++ Server Configuration .....	35
<b>Third Party Licenses .....</b>	<b>37</b>

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

ODBC is one of 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 3.80 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](http://msdn.microsoft.com/en-us/library/ms714562(vs.85).aspx)

## About SimbaEngine

SimbaEngine is a complete implementation of the ODBC specification, which provides a standard interface to which any ODBC enabled application can connect. 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 an executable file that will be accessed by common reporting applications and to access your data store when SimbaEngine executes an SQL statement. This executable file can be a Windows DLL, a Linux or Unix shared object, a stand-alone server, or some other form of executable. 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 UltraLight sample driver

The UltraLight driver is a sample DSI implementation of an ODBC driver, written in C++, which reads hard coded data. For demonstration purposes, the data is represented by a hard-coded table object called the Person table, which will always be returned if an executed query contains the word "SELECT". If the query does not contain the word "SELECT" then a row count of 12 rows will be returned.

The UltraLight driver helps to prototype a DSI implementation for SQL-based data store. You can also use it as the foundation for 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 customers.

Implementation, begins with the creation of a DSIDriver class which is responsible for constructing a DSIEnvironment. This in turn is used to construct a connection object (DSIConnection implementation) which can then be used for constructing statements (DSIStatement implementations). This is summarized in [About the UltraLight sample driver](#).

|

Figure 1 – Core Component Implementation

The DSIStatement implementation is responsible for creating a DSIDataEngine object which in turn creates IQueryExecutor objects to execute queries and hold results (IResults), and DSIMetadataSource objects to return metadata information. This is summarized in [About the UltraLight sample driver](#):

|

Figure 2 – DataEngine Implementation

The final key part of the DSI implementation is to create the framework necessary to retrieve both data and metadata. A summary of this framework and the components implemented by the sample are shown in [About the UltraLight sample driver](#):

|

Figure 3 – Design pattern for a DSI implementation.

The IResult class is responsible for retrieving column data and maintaining a cursor across result rows.

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

The various “MetadataSource” classes provide a way for the calling framework to obtain metadata information.

## 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 UltraLight 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 such as naming the driver, setting the properties that configure the driver, and naming the XML error file and log files. The other areas of the code that you will modify are related to getting the data and metadata from

your data store. Since the UltraLight driver already has the classes and code to do this against its example data store (hard coded data), all you have to do is modify the existing code to make your driver work against your own data store.

# Day One – Windows Instructions

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

## Install SimbaEngine

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

1. Launch Microsoft Visual Studio.
2. Click **File > Open > Project or Solution**.
3. Navigate to `[INSTALLDIR]\SimbaEngineSDK\10.3\Examples\Source\UltraLight\Source` and then open the `UltraLight_vs201x.sln` file.  
The default `[INSTALLDIR]` is `C:\Simba Technologies`.
4. Click **Build > Configuration Manager** and make sure that the active solution configuration is "Debug\_MTDLL" and then click **Close**.
5. Click **Build > Build Solution** or press F7 to build the driver.

This will build the debug version of the driver and place it in the following location for 32-bit drivers:

`[INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\UltraLight\Bin\Win32\Debug_MTDLL`

Or will place it in this location for 64-bit drivers:

`[INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\UltraLight\Bin\x64\Debug_MTDLL`



# 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 or 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 connect to and configure your driver.

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 have to create similar registry keys.

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 1 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, you must use the 32-bit ODBC Data Source Administrator. You cannot access the 32-bit ODBC Data Source Administrator from the start menu or control panel on 64-bit Windows. Only the 64-bit ODBC Data Source Administrator is accessible from the start menu or control panel. On 64-bit Windows, to launch the 32-bit ODBC Data Source Administrator you must run `C:\WINDOWS\SysWOW64\odbcad32.exe`. See [ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit](#) on page 1 for details.
2. In the ODBC Data Source Administrator, click the **System DSN** tab.
3. Scroll through the list of System Data Sources, select **UltraLightDSII** and then click **Configure**. The Data Source Configuration window opens and displays the fields for the user ID, password, and language.
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\`.

Navigate to the folder that corresponds to your machine's architecture (amd64, ia64 or x86) and then click `odbc32.exe` to launch the ANSI version or click `odbc32w.exe` to launch the Unicode version.

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

2. In the ODBC Test tool, select **Conn > Full Connect**. The Full Connect window opens.
3. 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.

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

## 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.

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 won't be possible.

1. In your Windows Explorer window, copy the `[INSTALLDIR]\SimbaEngineSDK\10.3\Examples\Source\UltraLight` directory and paste it to the same location. This will create a new directory called `UltraLight - Copy`. Rename the directory to something that is meaningful to you. This will be the top-level directory for your new project and DSI implementation files. For the rest of this tutorial, when you see `<YourProjectName>` in the instructions, replace this with the name you choose for this directory which is also the name of your project.
2. Open your new directory, open the `Source` directory and rename the `UltraLight_vs201x.vcxproj` file in it to `<YourProjectName>.vcxproj` file where you replace `<YourProjectName>` with the name of your project. This will be the project file for your new ODBC driver.
3. Rename the `.sln` file. This new `<YourProjectName>.sln` file is the solution file for your new ODBC driver.
4. Using a text editor, open the project file (`.vcxproj`) and replace every instance of `UltraLightDSII` in the source code with the name of your new ODBC driver. Then save and close the file.
5. Using a text editor, open the solution file (`.sln`) and replace every instance of `UltraLightDSII` in the source code with the name of your new ODBC driver. In addition, references to the name of the project file must be updated to match the `<YourProjectName>.vcxproj` project file that you renamed. Then, save and close the file.

## Build your new driver

1. Launch Microsoft Visual Studio.
2. Click **File > Open > Project or Solution**.
3. Navigate to `[INSTALLDIR]\SimbaEngineSDK\10.3\Examples\Source\<YourProjectName>\Source` and then open the `<YourProjectName>.sln` file.

4. Click **Build > Configuration Manager** and make sure that the active solution configuration is "Debug\_MTDLL" and click **Close**.
5. Click **Build > Build Solution** or press F7 to build the driver.  
This will build the debug version of the driver and place it in the location:  
[INSTALLDIR]\SimbaEngineSDK\10.3\Examples\Source\<YourProjectName>\Bin\Win32\Debug\_MTDLL
6. When you build your new project, "TODO" messages appear in the Output window along with the build information.  
If the Output window is not displayed automatically, you can open it by selecting **Debug > Windows > Output**.

TODO #1: Construct driver singleton.

TODO #2: Set the driver properties.

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

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

TODO #5: Check Connection Settings.

TODO #6: Customize DriverPrompt Dialog.

TODO #7: Establish A Connection.

TODO #8: Create and return your Metadata Sources.

TODO #9: Prepare a Query.

TODO #10: Implement an IQueryExecutor.

TODO #11: Provide parameter information.

TODO #12: Implement Query Execution.

TODO #13: Implement your DSISimpleResultSet.

TODO #14: Register the ULMessages.xml file for handling by DSIMessageSource.

TODO #15: Set the vendor name, which will be prepended to error messages.

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

## 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\MyUltraLight\Source`.
2. For 32-bit Windows, open `SetupMyUltraLightDSII-32on32.reg`.  
  
For a 32-bit ODBC driver on 64-bit Windows, open `SetupMyUltraLightDSII-32on64.reg`.  
  
For a 64-bit ODBC driver on 64-bit Windows, open `SetupMyUltraLightDSII-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. Next, update the ODBC Data Sources section to add your new data source. Under the `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources]` section, change `MyUltraLightDSII="MyUltraLightDSIIDriver` to the name of your new data source and new driver. For example,  
`<YourProjectName>DSII"="<YourProjectName>DSIIDriver`.
5. Then, modify the data source definition for that data source. Change the line that says `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\MyUltraLightDSII]` so that it contains your new data source name. For example, `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\<YourProjectName>DSII]`.
6. Beside the line that starts with "Driver"= change the driver name to that of your new ODBC driver. For example, "Driver"="<YourProjectName>DSIIDriver".
7. Update the ODBC Drivers section to add your new driver. Under the `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers]` section, change `"MyUltraLightDSIIDriver"="Installed"` to match the name of your new driver. For example, `"<YourProjectName>DSIIDriver"="Installed"`.
8. Modify the driver definition for that driver. Change the line that says `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\MyUltraLightDSIIDriver]` so that it contains your new driver name. For example, `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\<YourProjectName>DSIIDriver]`
9. Beside the lines that starts with "Driver" and "Setup", update the path to the dll file for both.
10. Click **Edit > Find and Replace > Quick Replace**. Then, replace "UltraLight" in the whole file with the name of your new ODBC driver.
11. Click **Save** and close the file.
12. In the Registry Editor (`regedit.exe`), click **File > Import**, navigate to the registry file that you just modified and 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, you must use the 32-bit ODBC Data Source Administrator. You cannot access the 32-bit ODBC Data Source Administrator from the start menu or control panel on 64-bit Windows. Only the 64-bit ODBC Data Source Administrator is accessible from the start menu or control panel. On 64-bit Windows, to launch the 32-bit ODBC Data Source Administrator you must run `C:\WINDOWS\SysWOW64\odbcad32.exe`. See [ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit](#) on page 1 for details.
2. In the ODBC Data Source Administrator, click 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 `odbctc32.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 click **Debug > Attach to Process**.
3. In the **Attach to Process** window, select the **ODBC Test process** and click **Attach**. The process name will be either `odbc32.exe` or `odbct32w.exe`.
4. Add a breakpoint in `Main_Windows.cpp`, on the function `DSIDriverFactory()`. This function runs as soon as the Driver Manager loads the ODBC driver.
5. In the ODBC Test tool, select **Conn > Full Connect**. The Full Connect window opens.

6. Select your Data Source from the list of data sources and click **OK**.  
If you do not see your data source in the list, make sure that you are running the version of the ODBC Test tool that corresponds to the version of the data source that you created. In other words, if you created a 32-bit data source then you should be using the 32-bit version of the ODBC Test tool.
7. You should hit the breakpoint you created and focus should switch to Visual Studio.
8. To continue running the program, select **Debug > Continue**. The focus returns to the ODBC Test window.
9. Enter `Select * from ULResultSet` in ODBC Test and click the button on the toolbar with the exclamation icon. Then click the button beside it. This will output a simple result set.

## Summary – Day One

At this point, you have completed the following tasks:

- Install SimbaEngine and build the sample driver included with the SDK.
- Learn about the Windows ODBC Data Source Administrator, the creation of new Data Source names and the area of the Windows Registry where these settings are stored.
- Test the sample drivers using an ODBC-enabled application.
- Set up a new project directory where you will begin to modify one the sample drivers as the starting point for your new driver.

## Day Two

Today's goal is to customize the connector, enable logging and establish a connection to the data store. To accomplish this, check TODO items 1 to 7.

Remember that, when the project is built, you will see the TODO messages in the Output window. To rebuild the whole solution, select **Build > Rebuild Solution**. If it does not display, open the Output window by selecting **Debug > Windows > Output**. Double click the TODO relevant section of code.

## Construct a Connector Singleton

### TODO #1: Construct a Connector singleton

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

To construct the connector singleton:

1. Launch Microsoft Visual Studio.
2. Click **File > Open > Project/Solution**
3. Navigate to  
`[INSTALLDIR]\SimbaEngineSDK\10.0\Examples\Source\<YourProjectName>\Source`  
and then open the `<YourProjectName>_VS201x.vcproj` file.
4. Rebuild your solution and then double click the TODO #1 message section of code. The `Main_Windows.cpp` file opens.
5. Look at the `DSIDriverFactory()` implementation.
6. Specify the location that is used when reading driver settings from the registry. This change is related to rebranding. Locate the line of code where the `#define` directive specifies `DRIVER_WINDOWS_BRANDING`, and replace the value with something like `"Company\\Driver"` where `"Company"` is your company name and `"Driver"` is the name of your driver.

This step, like those in day one, is important to distinguish your driver from our sample and other drivers.

On Windows, this changes the registry node where the driver settings are read from, while on other platforms, this changes the name of the .ini file where the settings are read from. The `\Driver` or `\Server` suffix is added depending on configuration. On non-Windows platforms, this will be set to something like `"company.driver.ini"`.

7. Add the processing, if you are building a commercial connector.
8. Click **Save**.

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



# Set the Connector properties

## TODO #2: Set the driver properties

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

# Set the logging details

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

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

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

1. Double click the TODO #3 section of code.
2. Change the driver log's file name.
3. Double click the TODO #4 section of code.
4. The connections currently use the same log file as the driver, you may choose to have each connection create a separate log file. If so, change the code to create a `DSILog` with a unique log file name.
5. Click **Save All**.

By default, the SimbaEngine UltraLight Driver maintains a log file for the entire driver. If you require more fine grained logging, then consider one for all driver-based calls and one for each connection created as noted in step 4, above. For more information about how to enable logging, refer to the SimbaEngine Developer Guide

# Check the connection settings

TODO #5: Check Connection Settings.

When the Simba ODBC layer is given a connection string from an ODBC-enabled application, the Simba ODBC layer parses the connection string into key-value pairs. 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. Validating the correctness (eg. Passwords) is done later in the `Connect()` method.

For example, the connection string `DSN=UltraLight; UID=user` will be broken down into key-value pairs and passed through the `DSIConnSettingRequestMap` parameter. In this case that map 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 #5 message section of code.
2. The `UpdateConnectionSettings()` function should validate that the key-value pairs in `in_connectionSettings` are sufficient to create a connection, and any settings that are not present should be added to the `DSIConnSettingResponseMap` parameter.

The `VerifyRequiredSetting()` or `VerifyOptionalSetting()` utility functions can be used to perform this verification and will add missing settings to `DSIConnSettingResponseMap`. For example, the UltraLight driver verifies that the entries within `in_connectionSettings` are sufficient to create a connection, by using the following code:

```
VerifyRequiredSetting(UL_UID_KEY, in_connectionSettings, out_connectionSettings);
VerifyRequiredSetting(UL_PWD_KEY, in_connectionSettings, out_connectionSettings);
VerifyOptionalSetting(UL_LNG_KEY, in_connectionSettings, out_connectionSettings);
```

The UltraLight driver requires a user ID and password, and can optionally take in a language (not currently used).

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

## Customize the DriverPrompt Dialog

### TODO #6: Customize DriverPrompt Dialog.

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 the `DSIConnSettingResponseMap`, 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 the `DSIConnSettingResponseMap`.

`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:

- **in\_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.
- **io\_connectionSettings:** A connection settings map which is populated by the dialog with settings entered by the user.
- **in\_parentWindow:** The handle to the parent Window to make the prompt window a child of.
- **in\_promptType:** An enum specifying if only required fields are to be available, or if optional fields should be available as well. In the UltraLight driver, the language is an optional field.

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

On Linux, no such dialog is displayed by this implementation. Instead, the window handle and prompt enum are ignored while the connection settings parameter is populated with empty values for the user ID and password fields:

```
(io_connectionSettings)[UL_UID_KEY] = Variant(simba_wstring(""));
(io_connectionSettings)[UL_PWD_KEY] = Variant(simba_wstring(""));
```

Code will therefore need to be added on Linux to get these values from somewhere (e.g. a dialog box, configuration file, etc.).

This step may be deferred until later to speed up the initial development of your driver. To defer it, leave the TODO in place and modify the method to return false. Until you return and implement this, you will need to ensure that you always provide complete connection information in your connection string or DSN settings.

## Establish a connection

TODO #7: Establish A Connection.

Once `ULConnection::UpdateConnectionSettings()` returns `out_connectionSettings` 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. Use the obtained values (eg. hostname, username, password, etc.) to make a connection with your datasource by passing them to your relevant API or network protocol.

1. Double click the TODO #7 message section of code.
2. Look at the code that authenticates the user against your data store using the information provided within the `in_connectionSettings` parameter. The sample code uses the utility function: `GetRequiredSetting()`. Alternatively, if authentication fails, you can choose to throw an `ErrorException` seeded with `DIAG_INVALID_AUTH_SPEC`.

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 the 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 #8: Create and return your Metadata Sources.

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

1. Creating a data structure that holds the data relevant for your data store:  
`Constructor`
2. Navigating the structure on a row-by-row basis: `Move()`
3. Retrieving data: `GetMetadata()` (See [Data Retrieval](#), [Data Retrieval](#) for a brief overview of data retrieval). Each column in the metadata source will be represented by a `DSIOutputMetadataColumnTag` which is passed into `GetMetadata()`.

## Handle DSI\_TYPE\_INFO\_METADATA

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

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

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

SQL_BIGINT	SQL_BINARY	SQL_BIT
SQL_CHAR	SQL_DECIMAL	SQL_FLOAT
SQL_DOUBLE	SQL_INTEGER	SQL_LONGVARBINARY
SQL_LONGVARCHAR	SQL_LONGWVARCHAR	SQL_NUMERIC
SQL_REAL	SQL_SMALLINT	SQL_TINYINT
SQL_TYPE_DATE	SQL_TYPE_TIME	SQL_TYPE_TIMESTAMP
SQL_VARBINARY	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::InitializeData()`. Populate the `m_dataTypes` vector in this method, which defines the collection types that are supported along with their parameters.

## 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::MakeNewMetadataTable()` 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 `DSIMetadataTableId`, which does not correspond to these tables, `ULDataEngine::MakeNewMetadataTable()` 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.

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

For more information on the other metadata source types, please refer to the `DSIMetadataTableId.h` header file.

## 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.

### Prepare and execute a query

#### TODO #9: Prepare a Query.

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

`ULDataEngine::PopulateParameters()` will populate parameters as described below. In your implementation you would replace this with more sophisticated logic or pass the query to the data source for preparation.

Not all data sources support the notion of preparing a query to the extent that they will be able to have a query plan and produce all the resultant metadata. In these cases, the `DSII` should make a best effort to determine the number of required query parameters and whether the first result is a rowcount or result set. Precise metadata of the parameters and columns may improve how applications behave with the driver but is not strictly necessary if the data source can only make guesses at this point. Character types are often a safe guess as they support most conversions.

#### TODO #10: Implement an IQueryExecutor.

The `ULQueryExecutor` object returned by the `ULDataEngine::Prepare()` method is an implementation of `IQueryExecutor` which, as the name suggests, executes a query. After preparing a query, an application may execute it multiple times, in which case a single `IQueryExecutor` would be created by the prepare and would then be used for each execution.

The implementation of `ULQueryExecutor` simply checks if the query passed in contains a select statement or not by looking at the `in_isSelect` parameter. If `in_isSelect` is set then the constructor creates and adds a simple result set consisting of people's names to `m_results`. Otherwise, it creates and adds a simple row count.

In your implementation, the retrieval and storage of the result set can be moved out of this method and into `ULQueryExecutor::GetResults()`.



Note that `GetResults()` will be called before query execution to retrieve and inspect the result set's metadata. This is because ODBC allows applications to retrieve column metadata from a query before execution, although the metadata does not need to be accurate until after execution.

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

**TODO #11: Provide Parameter Information.**

`ULQueryExecutor::PopulateParameters()` method is where parameter information is 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 will only be called if `ULQueryExecutor::GetNumParams()` indicates that there is at least one parameter in the query and if the hosting application doesn't set `SQL_ATTR_ENABLE_AUTO_IPD` to false.

**TODO #12: Implement Query Execution**

The next step is to handle statement execution in `ULQueryExecutor::Execute()`. The sample implementation simply resets the results obtained in the constructor in preparation for the application to retrieve them. If the executor is handling a parameterized statement, then additional logic iterates through the input and copies it to the output for consumption by the calling application.

In the implementation, the `Execute()` method should begin by serializing parameters (stored in `in_inputParamSetter`) 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 `in_outputParamSetIter` parameter.

After this method exits, the calling framework will then invoke `ULQueryExecutor::GetResults()` to obtain the result set.

**TODO #13: Implement your DSISimpleResultSet**

The final step in returning data is to implement a `DSISimpleResultSet`. The sample contains an implementation called `ULResultSet` 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, `ULResultSet::MoveToNextRow()` simply increments an row iterator so this should be replaced in your implementation with code that delegates this to the data source.

The `RetrieveData()` method is where column data is retrieved, so this should also be modified to extract data from the data source. (See [Data Retrieval](#), [Data Retrieval](#) for a brief overview of data retrieval)

## Day Five

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

## Configure error messages

### **TODO #14: Register Messages xml file for handling by DSIMessageSource.**

---

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

1. Rename the `ULMessages.xml` file to something appropriate to your data store.
2. Double click the TODO #14 message section of code.
3. Update the line associated with the TODO to match the new name of the file.
4. Open the renamed file and change all instances of the following items:
  - The letters `UL` to a two letter abbreviation of your choice in each `<Error>` element
  - The word `UltraLight` to a name relating to your driver
5. When you are done, you should revisit each exception thrown within your DSI implementation and change the parameters to match as well. This will rebrand your converted SimbaEngine UltraLight Driver for your organization.

### **TODO #15: 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 #15 message section of code.
2. Set the vendor name as shown in the commented code.

## Finishing touches

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

1. Rename all files and classes in the project to have the two-letter abbreviation chosen as part of TODO #14.
2. Create a driver configuration dialog. This dialog is presented to the user when they use the ODBC Data Source Administrator to create a new ODBC DSN or configure an existing one. The C++ SimbaEngine UltraLight Driver project contains an example ODBC

configuration dialog that you can look at, as an example. You can find the source in the SimbaEngine UltraLight Driver Visual Studio project.

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

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

On Linux and UNIX platforms, it is also possible to create a driver configuration dialog although our UltraLight sample driver for those platforms does not include a sample implementation.

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 UltraLight sample driver. Now, you have a read-only driver that connects to your data store.

## Reference

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

### 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 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 each registry key, and a **Locale** to specify the language. The keys that are relevant to the C++ examples discussed in this document are:

- **UltraLightDSII** which must include the following string values:
  - **Driver:** `UltraLightDSIIDriver`
  - **Description:** `Sample 32-bit SimbaEngine UltraLight 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 must include the following string values:
  - **UltraLightDSII:** `UltraLightDSIIDriver`

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

- **UltraLightDSIIDriver** which includes the following key names and values:
  - **Driver:** `[INSTALLDIR]\Examples\Builds\Bin\Win32\Release_MTDLL\UltraLightDSII_MTDLL.dll`
  - **Setup:** `[INSTALLDIR]\Examples\Builds\Bin\Win32\Release_MTDLL\UltraLightDSII_MTDLL.dll`
  - **Description:** `Sample 32-bit SimbaEngine UltraLight 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 string values:
  - **UltraLightDSIIDriver:** `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 three 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 three keys that are relevant to the C++ examples discussed in this document are:

- **UltraLightDSII** which must include the following string values:
  - **Driver:** `UltraLightDSIIDriver`
  - **Description:** `Sample 64-bit SimbaEngine UltraLight 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 must include the following string values:
  - **UltraLightDSII:** `UltraLightDSIIDriver`

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

- **UltraLightDSIIDriver** which includes the following key names and values:
  - **Driver:** (points to the driver DLL) `[INSTALLDIR]\Examples\Builds\Bin\x64\Release_MTDLL\UltraLightDSII_MTDLL.dll`
  - **Setup:** (points to the configuration DLL, which in most cases, is embedded in the driver DLL) `[INSTALLDIR]\Examples\Builds\Bin\x64\Release_MTDLL\UltraLightDSII_MTDLL.dll`
  - **Description:** `Sample 64-bit SimbaEngine UltraLight 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 string values:
  - **UltraLightDSIIDriver:** `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. `DSISimpleResultSet::RetrieveData()`

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

In addition, both methods accept the following three parameters:

1. `in_data`

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

The type of this parameter is governed by the metadata for the column that is returned by the class. Thus, if you set the SQL Type of column 1 in `DSISimpleResultSet::InitializeColumns()` to `SQL_INTEGER`, then when `DSISimpleResultSet::RetrieveData()` is called for column 1, you will be passed a `SQLData` that wraps a `simba_int32` (or `simba_uint32` if unsigned) data type. For `MetadataSource`, the type is associated with the column tag (see `DSIOutputMetadataColumnTag.h`).

For character or binary data you must call `SetLength()` before calling `GetBuffer()`. Not doing so may result in a heap-violation. See `ULResultSet.cpp` for an example on how to handle character or binary data.

2. `in_offset`

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

3. `in_maxSize`

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

## SqlData types

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

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

## Fixed length types

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

`SQL_BIT`



SQL\_DATE  
SQL\_DECIMAL  
SQL\_DOUBLE  
SQL\_GUID  
SQL\_FLOAT  
SQL\_INTEGER  
SQL\_INTERVAL\_DAY  
SQL\_INTERVAL\_DAY\_TO\_HOUR  
SQL\_INTERVAL\_DAY\_TO\_MINUTE  
SQL\_INTERVAL\_DAY\_TO\_SECOND  
SQL\_INTERVAL\_HOUR  
SQL\_INTERVAL\_HOUR\_TO\_MINUTE  
SQL\_INTERVAL\_HOUR\_TO\_SECOND  
SQL\_INTERVAL\_MINUTE  
SQL\_INTERVAL\_MINUTE\_TO\_SECOND  
SQL\_INTERVAL\_MONTH  
SQL\_INTERVAL\_SECOND  
SQL\_INTERVAL\_YEAR  
SQL\_INTERVAL\_YEAR\_TO\_MONTH  
SQL\_NUMERIC  
SQL\_REAL  
SQL\_SBIGINT  
SQL\_SINTEGER  
SQL\_SMALLINT  
SQL\_SSMALLINT  
SQL\_STINYINT  
SQL\_TINYINT  
SQL\_TIME  
SQL\_TIMESTAMP  
SQL\_TYPE\_DATE  
SQL\_TYPE\_TIME  
SQL\_TYPE\_TIMESTAMP  
SQL\_UBIGINT

SQL\_INTEGER

SQL\_USMALLINT

SQL\_UTINYINT

### More information on Date, Time and DateTime types

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

### Simple Fixed-Length Data Example

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

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

## Variable Length Types

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

`SQL_BINARY`

`SQL_CHAR`

`SQL_LONGVARBINARY`

`SQL_LONGVARCHAR`

`SQL_VARBINARY`

`SQL_VARCHAR`

`SQL_WCHAR`

`SQL_WLONGVARCHAR`

`SQL_WVARCHAR`

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

## Simple Variable-Length Data Example

The `SQL_CHAR` example below illustrates how the type utilities might be used while the `SQL_VARCHAR` example shows a simple example using `memcpy`. In practise, `SQL_CHAR`, `SQL_VARCHAR` and `SQL_LONGVARCHAR` will not need separate cases to handle them and there will also be other considerations such as having to deal with offsets into the data.

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

## NULL Values

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

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

# Appendix D: C++ Server Configuration

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



**Important:** The information in this section only applies if you are using 32-Bit Windows. If you are using 64-bit Windows (with either 32-bit or 64-bit applications), the file paths must be configured appropriately. Please see [Appendix B: Windows Registry 32-Bit vs. 64-Bit](#) for details.

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

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

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

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

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

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

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

# 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](mailto: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([eyay@cryptsoft.com](mailto:eyay@cryptsoft.com)). This product includes software written by Tim Hudson ([tjh@cryptsoft.com](mailto:tjh@cryptsoft.com)).

Original SSLeay License

Copyright (C) 1995-1998 Eric Young ([eyay@cryptsoft.com](mailto:eyay@cryptsoft.com))

All rights reserved.

This package is an SSL implementation written by Eric Young ([eyay@cryptsoft.com](mailto:eyay@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](mailto: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 routines 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 NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

Stringencoders License

Copyright 2005, 2006, 2007

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

All rights reserved.

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

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

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

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

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

This is the standard "new" BSD license:

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

dtoa License

The author of this software is David M. Gay.

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

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

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