



## **SimbaEngine SDK 9.0**

# **Build a C++ ODBC Driver in 5 Days**

**2012-01-31**

**Simba Technologies Inc.**



Copyright © Simba Technologies Inc. All Rights Reserved.

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this publication, or the software it describes, may be reproduced, transmitted, transcribed, stored in a retrieval system, decompiled, disassembled, reverse-engineered, or translated into any language in any form by any means for any purpose without the express written permission of Simba Technologies Inc.

### Simba Trademarks

Simba, the Simba logo, SimbaEngine, SimbaEngine C/S, SimbaClient, SimbaD20, SimbaEngine SDK and SimbaODBC are registered trademarks of Simba Technologies Inc. All other trademarks and/or servicemarks are the property of their respective owners.

### Simba Technologies Inc.

938 West 8<sup>th</sup> Avenue  
Vancouver, BC Canada  
V5Z 1E5

Tel. +1.604.633.0008  
Fax. +1.604.633.0004

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

Printed in Canada

## Third Party Trademarks

ICU License – ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995–2010 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

Copyright (c) 1998–2008 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.

### 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."

## Table of Contents

1	Before You Begin.....	1
1.1	Who Should Read this Manual.....	1
1.2	Conventions Used in this Manual.....	1
1.3	For More Information.....	2
1.4	Contact Us.....	3
2	Getting Started.....	4
2.1	What You’re Working Toward.....	4
2.2	How You Will Get There.....	5
2.3	What’s Included in the SDK.....	7
2.3.1	The SimbaEngine SDK Folder Structure after a Windows Installation.....	8
2.3.2	Libraries Included With the SDK.....	9
2.3.3	Registry Entries for Drivers and Data Source Names.....	10
3	Build an ODBC Driver in Five Days.....	13
3.1	Day One.....	14
3.2	Day Two.....	20
3.3	Day Three.....	21
3.4	Day Four.....	23
3.5	Day Five.....	25
Appendix A:	ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit.....	27
Appendix B:	Windows Registry 32-Bit vs. 64-Bit.....	28
Appendix C:	Data Retrieval.....	32
Appendix D:	How to Add Schema Support.....	37
Appendix E:	C++ Server Configuration.....	38

## Table of Figures

Figure 1: High level view of SimbaEngine .....	4
Figure 2: Design pattern for a DSI implementation.....	6
Figure 3: The structure of the folders installed by SimbaEngine SDK.....	9

# 1 Before You Begin

This guide provides condensed information to walk you through creating a custom ODBC driver with the SimbaEngine SDK, using C++ as your development environment.

There are five sample drivers provided with the SimbaEngine SDK:

1. The activities in this guide use the Quickstart sample driver (which accesses the most basic type of text-based data source) as a base from which you will work.
2. A similar pattern of activities can be followed using the Codebase sample driver (which provides implemented examples of more advanced features such as bookmarks and Collaborative Query Execution). For further details on the Codebase example driver, please see the SimbaEngine SDK Developer Guide.
3. The DotNetQuickstart sample driver is the C# version of the Quickstart driver. There is a separate variation of this document to cover that driver, titled: *Build a C# ODBC Driver in 5 Days*.
4. The JavaQuickstart sample driver is the Java version of the Quickstart driver. There is a separate variation of this document to cover that driver, titled: *Build a Java ODBC Driver in 5 Days*.
5. The UltraLight sample driver illustrates a connection to a database that already supports SQL and therefore does not require the SQLEngine component.

## 1.1 Who Should Read this Manual

This guide assumes you have a general understanding of ODBC architecture and have access to the Microsoft ODBC Software Development Kit.

This document also assumes you have a working understanding of modern database principles and terminology, Microsoft's Visual Studio development environment, the C++ development language, object orientated principals in general, and your development environment.

## 1.2 Conventions Used in this Manual

This document is primarily focused on a Windows-based development environment. However, at selected key points, significant differences relating to Linux/Unix environments are highlighted in blue-tinted text blocks like this one.

Directories, files, and parameter names appear in italics. For example:

The libraries containing the data access components you need are in the *C:\Simba Technologies\SimbaEngineSDK\9.0\DataAccessComponents* folder.

Computer input and output, such as sample listings, messages that appear on your screen, and commands or statements that you are instructed to type, appear in Courier typeface. For example:

```
SQLDriverConnect returned: SQL_ERROR=-1.
```

Function names, SQL keywords, and program names appear in narrow bold type when described in text. For example:

Implement the constructor **CustomerDSIIConnection::CustomerDSIIConnection**

The full path of the installation directory for the SDK may vary depending on your system. In this document, we will represent it as [INSTALL\_DIRECTORY], which defaults as follows (note that “9.0” is the current release and this part of the path will change with each release of the SDK):

- Windows platforms:  
C:\Simba Technologies\SimbaEngineSDK\9.0
- Linux/Unix/MacOSX platforms:  
<theUntarDirectory>/Simba/SimbaEngineSDK/9.0

## 1.3 For More Information

The following documentation is available for the SimbaEngine SDK:

- **SimbaEngine SDK Developer Guide:** Detailed information on how to work with the SDK to develop an ODBC/JDBC/ADO.NET driver for virtually any data store.
- **Build a C++ ODBC Driver in 5 Days:** (*this document*) Condensed information to walk you through the process of creating a custom ODBC driver with the SimbaEngine SDK, using C++ as your development environment.
- **Build a C# ODBC Driver in 5 Days:** Condensed information to walk you through the process of creating a custom ODBC driver with the SimbaEngine SDK, using C# as your development environment.
- **Build a Java ODBC Driver in 5 Days:** Condensed information to walk you through the process of creating a custom ODBC driver with the SimbaEngine SDK, using Java as your development environment.
- **Build a JDBC Driver in 5 Days:** Condensed information to walk you through the process of creating a custom Type 4 JDBC driver with the SimbaEngine SDK, using Java as your development environment.
- **Build an ADO.NET Provider in 5 Days:** Condensed information to walk you through the process of creating a custom ADO.NET Data Provider with the SimbaEngine SDK, using C# as your development environment.

- **SimbaEngine DSI API Reference Guide:** Detailed information about function parameters, return types and error and message codes.
- **SimbaClientServer Users Guide:** Detailed information explaining the creation, installation, configuration, and administration of the server and client components of the SimbaEngine SDK.
- **SimbaEngine SDK Release Notes:** Information about incremental changes introduced in a particular release of the SimbaEngine SDK.

For complete information on the ODBC 3.5 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)

## 1.4 Contact Us

Simba support is available from 9:00 AM to 5:00 PM Pacific Time, Monday to Friday. Send an e-mail to [support@simba.com](mailto:support@simba.com), or dial 604-633-0008 and select 3.

## 2 Getting Started

This document describes the steps required to build a prototype ODBC driver on Windows using the SimbaEngine SDK to access your data store. We have spaced the steps out over five days, but you are likely to finish much sooner than that.

The SimbaEngine SDK is a complete implementation of the ODBC specification, which provides a standard interface that any ODBC enabled application can connect to. The libraries of the SDK hide all of 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 primitive operations needed to access a data store. As an SDK developer, you will create an implementation of a DSI (also known as a DSI Implementation or DSII – highlighted in green in Figure 1) that will access your particular data source.

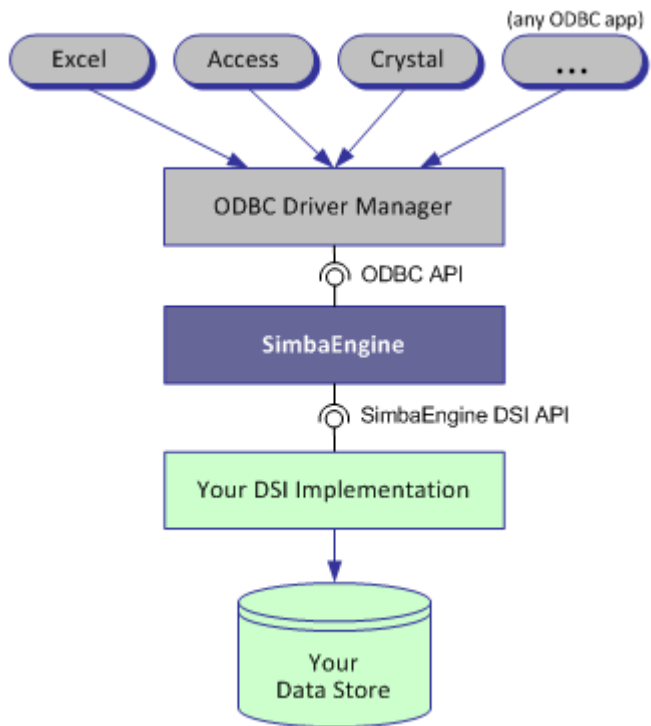


Figure 1: High level view of SimbaEngine

The sample projects provided with the SDK are functioning DSI Implementations that you can copy and modify for your own purposes. The Quickstart example that is the primary subject of this document connects to a simple text file. The Codebase example connects to a proprietary, non-SQL database. Following the steps in this guide, you will change the Quickstart example driver so that it accesses your own data store instead of the example data store included with the SDK.

### 2.1 What You’re Working Toward

The SimbaEngine libraries allow you to create data access solutions that connect to SQL or non-SQL data sources that are either local or remote to the end users’ computers. All of the possible SimbaEngine implementation architectures are discussed in more detail in the SimbaEngine SDK Developer Guide, but they are explained briefly here for context.

If you plan for your users to connect to your data source locally, you will compile and link your driver as a DLL, then install and register that on each computer containing a data source. If you plan to connect your users to a network-based data source, you will link your driver with the Simba Client/Server libraries (with no changes to your DSI code) to create a stand-alone SimbaServer executable that will run on your database server. You will then distribute the generic SimbaClient to individual users. The prototype that you will build by following this document accesses a local data store. Please see the SimbaClientServer User Guide for more details on remote database configurations.

Simba SQLEngine is part of the SDK as well, providing data access extensions to the DSI API for non-SQL data sources. For SQL-capable data sources, SQLEngine is simply not used and the SQL is passed through directly to your database. The prototype you will build by following this document accesses a non-SQL data source and thus uses SQLEngine.

On Linux or Unix systems, the final deliverable is either a shared object (equivalent to a DLL on Windows) or a stand-alone executable server.

## 2.2 How You Will Get There

In the SimbaEngine Quickstart driver code we have highlighted the areas you need to change by adding `#pragmas` that include the keyword “TODO” so you can find them, along with a short explanatory message. When you compile the code, these `#pragmas` show up in the output window. There are ten areas in the code highlighted with `TODOs` and this document will walk you through each of them.

Of the ten areas of the code that you need to modify, eight changes are for productization rather than actually connecting your data store to Simba SQLEngine. These are things like naming the driver, setting the properties that configure the driver, and naming the XML error file and log files. In other words, these are not complicated tasks.

The other two areas of the code that you will modify are primarily concerned with getting the data and metadata from your data store into the Simba SQLEngine. Since the Simba Quickstart driver already has the classes and code to do this against the example data store, all you have to do is modify what already exists and your driver will begin to work against your own data store.

The simplicity of the DSI API also helps you because there is order and symmetry to the way it works. The UML diagram below shows the design pattern to look for. Almost all DSI implementations wind up with a similar pattern. Look for the circular pattern of class relationships headed by `IResult` and anchored by your `Utilities` classes. Simba Quickstart Driver has the utilities class called `QUtilities`. If your resultant DSI implementation design looks like this, you are on the right track.

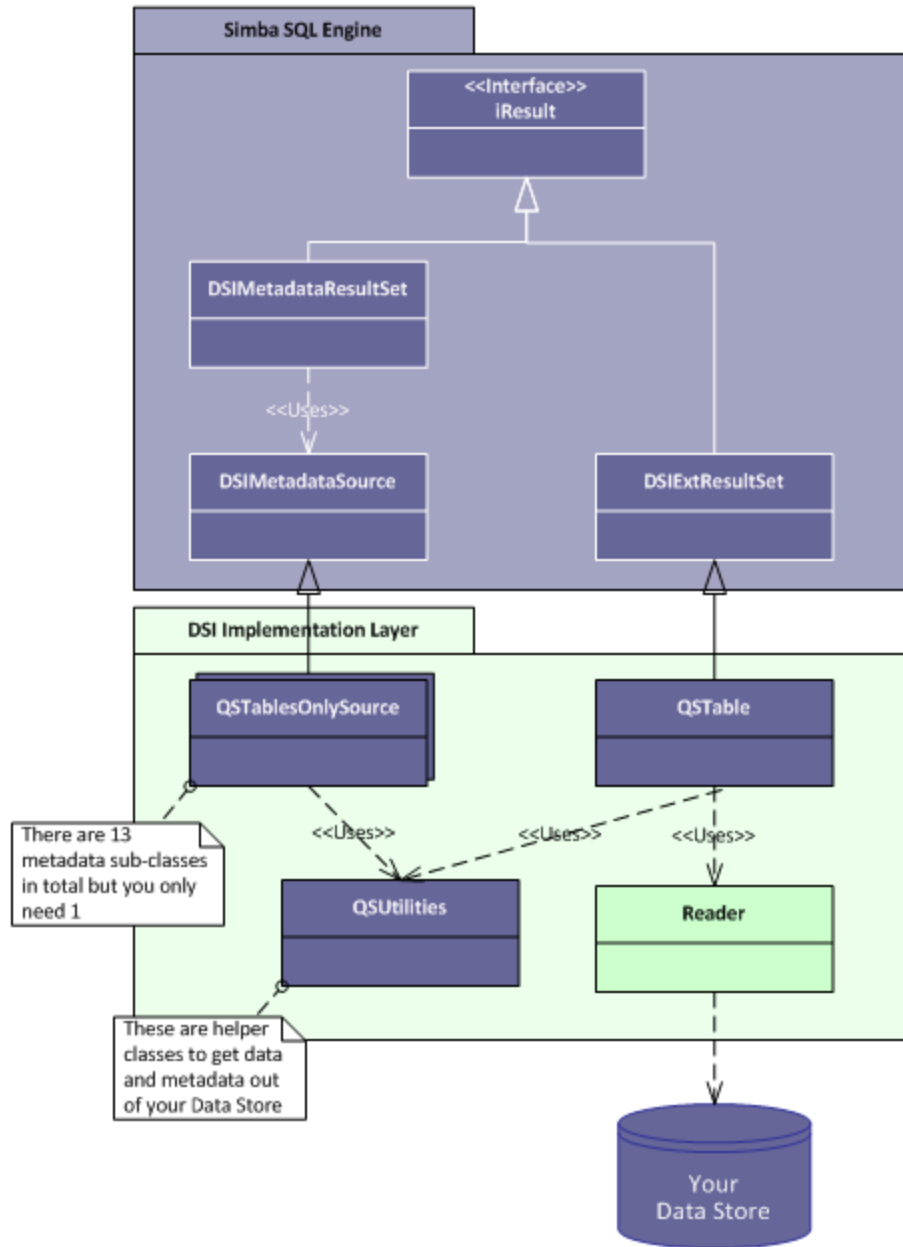


Figure 2: Design pattern for a DSI implementation.

Implementing data retrieval is straightforward. 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. Implementing metadata access is a bit more complicated, but it is not as bad as it looks. There are 15 sub-classes of MetadataSource that you can implement, but as a starting point, to make your driver work properly with Microsoft Excel you only need to implement the type information MetadataSource and the DSIMetadataHelper and return NULL for the rest.

## 2.3 What's Included in the SDK

Figure 3 on the next two pages shows the structure of all of the folders extracted to your computer by the installer. The default `[INSTALL_DIRECTORY]` is:

- Windows platforms:  
`C:\Simba Technologies\SimbaEngineSDK\9.0`
- Linux/Unix/MacOSX platforms:  
`<theUntarDirectory>/Simba/SimbaEngineSDK/9.0`

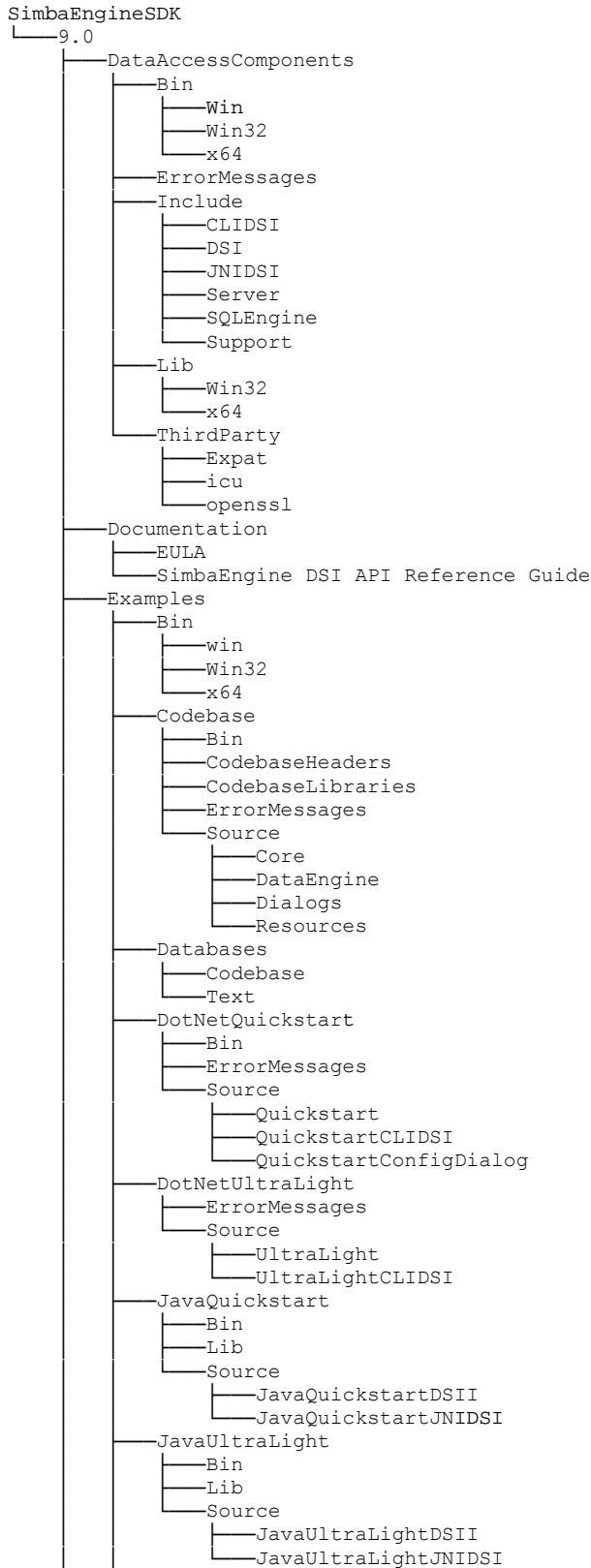
Note: The exact install directory tree structure for Linux/Unix/MacOSX is slightly different than the Windows tree shown below, but the general layout is similar.

Notice in the diagram below:

- `[INSTALL_DIRECTORY]\DataAccessComponents:`
  - `\Bin` sub-folder: The .Net libraries and SimbaClient.dll are found here.
  - `\Lib` sub-folder: The C++ libraries, Java libraries, and SimbaJDBCClient.jar are found here.
- `[INSTALL_DIRECTORY]\Examples:` Sub-folders containing all the source code and project files to create complete drivers that will work against provided sample data. The purpose of these examples is to provide complete solutions both so you can see how your custom solution will look when you are done, and actually provide a starting point for your driver.
  - `\Bin` sub-folder: Pre-built binaries of the sample drivers for immediate use and reference.
  - `\Databases` sub-folder: Sample databases to test the example projects against.
- `[INSTALL_DIRECTORY]\Documentation:` The developer and other guides.
- `[INSTALL_DIRECTORY]\Setup:`
  - Windows: Registry setup files for 64-bit installed example drivers as well as SimbaClient and SimbaServer example installations.
  - Linux/Unix/MacOSX: Example INI files.
- `[INSTALL_DIRECTORY]\SSLCertificates:` Simba self-signed example SSL certificates.

### 2.3.1 The SimbaEngine SDK Folder Structure after a Windows Installation

[INSTALL\_DIRECTORY] - defaults on Windows to C:\Simba Technologies\SimbaEngineSDK\9.0\



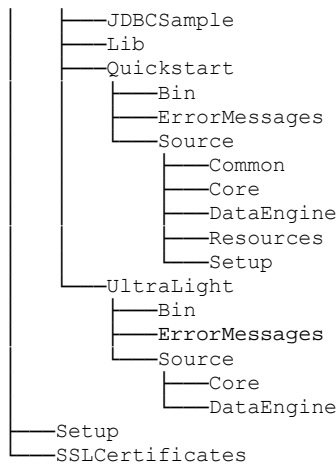


Figure 3: The structure of the folders installed by SimbaEngine SDK.

### 2.3.2 Libraries Included With the SDK

The SDK includes libraries for developing an ODBC driver using C++, Java, or C#. These libraries are in the `\Bin` sub-folder (.Net) and `\Lib` sub-folder (C++ and Java). The debug versions of the libraries have code optimization turned off and debugging information included in the linkable objects. The release versions have code optimizations turned on and no debugging information included. You can build your code with the same compiler options to create either debug or release executables. Typically, you will build the debug versions until you are ready to test performance or release candidates. Very occasionally, there are differences in the operation of the debug and release compiled versions of code. This mostly shows up in memory operations because compiled debug code is typically more tolerant of memory problems, such as buffer overruns, than is release code.

#### C++ Libraries

There are 32-bit (`\Lib\Win32`) and 64-bit (`\Lib\x64`) versions of the C++ libraries, allowing you to compile your code and create executable versions for each platform. Also, there are four linking options available for the run-time libraries—there are two statically-linked (`\Debug` and `\Release`) and two dynamically-linked (`\Debug_MTDLL` and `\Release_MTDLL`) versions of each. The examples in this document reference the 32-bit, statically-linked, debug versions of the libraries.

#### .NET Libraries

The `\Bin` subfolder contains the Release version of .Net libraries. Both the 32 and 64-bit drivers can use the same .Net libraries, because they are built for any CPU architecture.

## Java Libraries

The `\Lib` sub-folder contains the Release version of Java libraries. Both the 32 and 64-bit drivers can use the same JAR files, because they are bitness agnostic.

### 2.3.3 Registry Entries for Drivers and Data Source Names

When you install the SimbaEngine SDK on Windows, a series of registry keys are created, one set for Data Source Names (DSNs) and the other set to identify driver locations. Your custom driver installer will eventually have to create similar registry keys, so the information in this section will help you understand the types of keys involved and how they are related.

#### 32-Bit Drivers on 32-Bit Windows

**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 on page 28 for details.

#### Data Source Names

The ODBC Driver Manager uses Data Source Name registry keys to connect your driver to your database. The Simba installer automatically creates a DSN registry key for each sample driver included with the SDK. These keys are created in `HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBC.INI` and each key includes three string values to define the location of the Driver, the database (DBF) that it will use and a Description to help you clearly identify each registry key. The three keys that are relevant to the C++ examples discussed in this document are:

- **QuickstartDSII** which includes the following string values:
  - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartDSII.dll`
  - **DBF:** `[INSTALL_DIRECTORY]\Examples\Databases\Text`
  - **Description:** Sample 32-bit SimbaEngine Quickstart DSII
- **CodebaseDSII** which includes the following string values:
  - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\CodebaseDSII.dll`
  - **DBF:** `[INSTALL_DIRECTORY]\Examples\Databases\Codebase`
  - **Description:** Sample 32-bit SimbaEngine Codebase DSII
- **UltraLightDSII** which includes the following string values:
  - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightDSII.dll`
  - **Description:** Sample 32-bit SimbaEngine UltraLight DSII

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

- **ODBC Data Sources** which includes the following string values:

- QuickstartDSII: *QuickstartDSIIDriver*
- CodebaseDSII: *CodebaseDSIIDriver*
- UltraLightDSII: *UltraLightDSIIDriver*

## Driver Locations

In addition to the DSN registry keys, the ODBC Driver Manager also requires registry keys to define each driver and its setup location. These keys are created in *HKEY\_LOCAL\_MACHINE/SOFTWARE/ODBC/ODBCINST.INI* and 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:

- QuickstartDSIIDriver which includes the following key names and values:
  - Driver: *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartDSII.dll*
  - Setup: *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartDSII.dll*
  - Description: Sample 32-bit SimbaEngine Quickstart DSII
- CodebaseDSIIDriver which includes the following key names and values:
  - Driver: *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\CodebaseDSII.dll*
  - Setup: *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\CodebaseDSII.dll*
  - Description: Sample 32-bit SimbaEngine Codebase DSII
- UltraLightDSIIDriver which includes the following key names and values:
  - Driver: *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightDSII.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 are also added to it:

- ODBC Drivers which includes the following string values:
  - QuickstartDSIIDriver: *Installed*
  - CodebaseDSIIDriver: *Installed*
  - UltraLightDSIIDriver: *Installed*

On Linux and UNIX the equivalent configuration is handled via *ODBC.INI* and *ODBCINST.INI* files. Samples of these files are provided in *[INSTALL\_DIRECTORY]\Setup* from which you can copy driver and DSN information to modify your existing *ODBC.INI* and *ODBCINST.INI* files. If these files do not already exist on your system, you can copy the samples to your *\$HOME* directory. For more details on how to access Data Sources under Linux/Unix/MacOSX, please refer to the SimbaEngine SDK Developer Guide.

## Creating Registry Keys for Your Own Driver

The registry keys discussed above point to pre-compiled versions of the example drivers and are immediately useable after installation so you can begin exploring right away. Later (see section

3.1, “Day One” on page 14), to install your new driver, you will create registry keys that point to your own driver and database. You will modify and run one of the .reg files found in the `Source` folder for each example project to create new entries in `ODBCINST.INI`. Then you will use the ODBC Data Source Administrator to create new DSNs in `ODBC.INI`.

**IMPORTANT:** If you are using 64-bit Windows with 32-bit applications, you will require special instructions to access the 32-Bit ODBC Data Source Administrator because it is not accessible from the start menu or control panel. Please see Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 27 for details.

### 3 Build an ODBC Driver in Five Days

SimbaEngine SDK ships with sample drivers that you can use for the basis of your own drivers. Over the course of the 5-day plan, you will modify the Quickstart sample driver so that it accesses your own data store instead of the example data files included with the SDK. Here is a quick overview of what you will be doing each day:

Day	Activities
One	<ul style="list-style-type: none"> <li>• Install the SimbaEngine SDK and build the sample drivers included with the SDK.</li> <li>• 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.</li> <li>• Test the sample drivers using an ODBC-enabled application.</li> <li>• Set up a new project directory where you will begin to modify one the sample drivers as the starting point for your new driver.</li> </ul>
Two	<ul style="list-style-type: none"> <li>• Set the driver properties to give your driver a name and possibly change the data encoding to match your data store.</li> <li>• Set the driver-wide and connection-wide logging level.</li> <li>• Modify the connection setting validation method to match the connection string needed for your data store.</li> <li>• Modify the connection authentication method to match the settings needed for your data store.</li> </ul>
Three	<ul style="list-style-type: none"> <li>• Modify the method used to create and return your Metadata Sources, which will support ODBC catalog functions.</li> <li>• Modify the method used to support SQLGetTypeInfo ODBC catalog function with the data types supported by your data store.</li> <li>• Modify the Metadata Helper class used to retrieve the identifying information for tables in your data store for the SQLTables and SQLColumns ODBC catalog functions.</li> </ul>
Four	<ul style="list-style-type: none"> <li>• Modify the method used to open a table defined within your data store.</li> <li>• Modify the method used to retrieve the column information for a table in your data store.</li> <li>• Modify the methods used to navigate through and retrieve data from your data store.</li> </ul>
Five	<ul style="list-style-type: none"> <li>• Register your error message file.</li> <li>• Modify the parameters of the exceptions thrown by your driver to match as well.</li> <li>• Rename the files and classes to match the same two-letter abbreviation chosen for your error messages.</li> <li>• Create a driver configuration dialog (by modifying the source of the example ODBC configuration dialog provided).</li> </ul>

## 3.1 Day One

Today's task is to set up and test 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.

### Initial Set Up

Start by installing the SDK and running the example drivers:

Install the SimbaEngine SDK using the setup executable for your version of Visual Studio – run this program and follow the installer's instructions. The default locations and resulting folders are shown in

1. Figure 3, on page 7. **Note: If you have previously installed a version of the SimbaEngine SDK, uninstall it before installing the new one.**
2. If you have not already done so, take some time to read section 2 of this guide, "Getting Started" beginning on page 4. It contains detailed information on the content of the SDK and introduces you to the architecture of the SimbaEngine SDK solutions.
3. Using `regedit.exe`, examine the DSNs that were created by the installer. See section 2.3.3, "Registry Entries for Drivers and Data Source Names" on page 10 for details of what to look for. Keep `regedit` open for use in a later step.
4. You are now ready to build, test and verify that the example drivers are working. You can use any ODBC application for testing, such as MS Access, MS Excel or `ODBCTest32.exe` (see the SimbaEngine SDK Developer Guide for more information on `ODBCTest`).

On Linux and UNIX platforms, SimbaEngine is provided as a single file consisting of the `SimbaEngineSDK*.tar.gz` file, a tar format archive that has been compressed using the `gzip` tool (where the "\*" represents a string of alphanumeric characters that represent the build number and platform of the kit). On these platforms, the installation steps are as follows:

1. Open a command prompt and change to a directory where you would like to install SimbaEngine.
2. To uncompress `SimbaEngineSDK*.tar.gz`, enter:  

```
gzip -d SimbaEngineSDK*.tar.gz
```

This will extract the file `SimbaEngineSDK*.tar`
3. To install SimbaEngine SDK enter:  

```
tar -xvf SimbaEngineSDK*.tar
```

### Build and Test the Example Drivers

1. The source for the example ODBC drivers is located in the `[INSTALL_DIRECTORY]\Examples` folder. These instructions assume you will be working with the 32-bit version of the Visual Studio 2008 `Quickstart` Driver example on a 32-bit Windows operating system.

2. Open the `QuickstartDSII_net2008.vcproj` project file, located in `[INSTALL_DIRECTORY]\Examples\Quickstart\Source`.
3. From the Main Menu select Build->Configuration Manager and make sure that the Active solution configuration is Debug. Now select Build->Build Solution (F7) to build the driver. This will build the debug version of the driver and place it in the location `[INSTALL_DIRECTORY]\Examples\Quickstart\Bin\Win32\Debug`.

On Linux and UNIX platforms, the sample drivers include makefiles instead of Visual Studio solution files. On these platforms, the process to build each of the sample drivers is similar. Using the SimbaEngine Quickstart sample driver on 32-bit Linux as an example, the steps are as follows:

1. Open up a command prompt and change to the `[INSTALL_DIRECTORY]\Examples\Quickstart\Makefiles` directory.
2. Set the `SIMBAENGINE_DIR` environment variable:
 

```
export
SIMBAENGINE_DIR=/<theUntarDirectory>/SimbaEngineSDK/9.0/DataAccessCo
mponents
```
3. Run the makefile for the debug target. Note that other options may be specified on the commandline, and are dependent on the platform and target (for more information about the options and build configurations, please refer to the SimbaEngine SDK Developer Guide):
 

```
make -f Quickstart.mak debug
```

4. Using any text editor, edit the appropriate registry (.reg) file located in the same folder noted in step 2 (there are three registry files: one for 32-bit Windows, one for a 32-bit ODBC driver on 64-bit Windows, and one for a 64-bit ODBC driver on 64-bit Windows. For this example, you will be using `SetupMyQuickstartDSII-32on32.reg`. For more information about 32-bit versus 64-bit drivers, please see Appendix B: Windows Registry 32-Bit vs. 64-Bit on page 28).

For this Day One example, the only change you need to make is to replace `[INSTALL_DIRECTORY]` with the path to where the samples were installed (see the note at the bottom of

Figure 3, on page 9 if this is not obvious). Take note that you must enter double backslashes in the folder path or the entry will not be created. Run this file and it will update your Windows registry to register the combined ODBC driver and configuration DLL you have just created (note that we have combined the DSN configuration functionality and the ODBC driver into one DLL for convenience. You can separate these functions into different DLLs if you want when you customize your own driver).

5. Run the Windows ODBC Data Source Administrator. To do this, open the Control Panel, select Administrative Tools, and then select Data Sources (ODBC). Note: Administrative Tools is found under System and Security if your Control Panel is set to view by category.

**IMPORTANT:** If you are using 64-bit Windows with 32-bit applications, you will require special instructions to access the 32-bit ODBC Data Source Administrator because it is not accessible from the start menu or control panel. Please see Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 27 for details.

In the System DSN tab, click on Add. This will open the Create New Data Source dialog. Select MyQuickstartDSIIDriver and click on Finish. This will bring up the Quickstart Data Source Configuration dialog from the driver you just created. Give your new DSN a name. You can give it a description as well if you want. Then click on Browse and navigate to *[INSTALL\_DIRECTORY]\Examples\Databases\Text*. This is where the Tabbed Unicode database files which the Quickstart sample driver reads are stored. Click on OK to create your new ODBC DSN.

On Linux and UNIX platforms, to configure the Data Sources for your driver you will need to edit the ODBC.INI and ODBCINST.INI files. (Note: for more detailed information, please refer to the SimbaEngine SDK Developer Guide). Assuming that the .odbc.ini and .odbcinst.ini files exist in your \$HOME directory:

1. Edit the .odbc.ini file and add:

```
[ODBC Data Sources]
MyQuickstartDSII=MyQuickstartDSIIDriver

[MyQuickstartDSII]
Description=Sample 32-bit SimbaEngine Quickstart DSII
DBF=/

```

2. Edit the .odbcinst.ini file and add:

```
[ODBC Drivers]
MyQuickstartDSIIDriver=Installed

[MyQuickstartDSIIDriver]
Driver=/

```

3. Edit the .simba.quickstart.ini file and set the ODBCInstLib to the absolute path of the ODBCInst library for the Driver Manager that you are using. For example, for the iODBC Driver Manager this would be

```
ODBCInstLib=<driver manager dir>/lib/libiodbcinst.so (notice the 'i' after the
lib) and for unixODBC this would be
ODBCInstLib=<driver manager dir>/lib/libodbcinst.so
```

6. Open any ODBC-enabled application (e.g. ODBC Test) and, with your Quickstart driver solution open, attach Visual Studio to its process. Do this by selecting Debug->Attach to Process... from the Main Menu and clicking on the ODBC application you are using.
7. You should now be able to place breakpoints anywhere in the SimbaEngine Quickstart DSI implementation. A good breakpoint to start with is in `Main_Windows.cpp`, on the function `DSIDriverFactory()`. This function runs as soon as the Driver Manager loads the ODBC driver.
8. Execute an ODBC operation within the application. You should hit the breakpoint you created in the previous step and focus should switch to Visual Studio.

On Linux and UNIX platforms, to test the ODBC driver you first need to make sure that you have a Driver Manager installed. For more detailed information on Driver Managers and testing, please refer to the SimbaEngine SDK Developer Guide. Using the iODBC Driver Manager as an example, you can use a debugger such as GDB with a test utility such as `iodbctest`.

When using GDB with an application, you need to be aware that your ODBC driver is not actually loaded until the application is running and you make a connection. This means that you can set breakpoints either before it is loaded or after (depending on which breakpoint you want to hit). There are a number of ways to set breakpoints using GDB, but you should be able to set breakpoints using the line number or function name. For more information, please refer to your GDB documentation.

You should be able to place breakpoints anywhere in the SimbaEngine Quickstart DSI implementation. A good breakpoint to start with is at `Main_Unix.cpp` `DSIDriverFactory()`. This function runs as soon as the Driver Manager loads the ODBC driver.

To be able to run the ODBC driver, you will also need to add the correct ICU libraries (according to platform) to the `LD_LIBRARY_PATH`:

```
export
LD_LIBRARY_PATH=$SIMBAENGINE_DIR/ThirdParty/icu/Linux_x86/lib:$LD_LIBRARY_PATH
```

To test your driver and hit your breakpoint, you should be able to enter the following:

```
gdb iodbctest
(gdb) file
<example_dir>/Examples/SimbaEngineQuickstart/Bin/Linux_x86/libQuickstart_debug.so
(gdb) break Main_Unix.cpp:26
(gdb) run DSN=MyQuickstartDSII
```

9. Perform the same steps with the Simba Engine Codebase Example Driver. The differences in the steps should be obvious, but in summary they are:

- At step 2, you will be opening the `CodebaseDSII_net2008.vcproj`, project file, located in `[INSTALL_DIRECTORY]\Examples\Codebase\Source`.
- At step 4, you will be using the `SetupMyCodebaseDSII-32on32.reg` registry configuration file.
- At step 6, you will select `MyCodebaseDSIIDriver` and navigate to `C:\Simba Technologies\SimbaEngineSDK\9.0\Examples\Databases\Codebase`.

On Linux and UNIX platforms, please note that the SimbaEngine Codebase Example Driver is not available.

If there were no problems with the example drivers you built, you are now ready to set up a development project to build your own ODBC driver.

## Setting Up the Project

1. Copy the `Quickstart` directory and paste it to the same location. This will create a new directory called "`Quickstart - Copy`". Rename the directory to something that is meaningful to you. This will be the top-level directory for your new project and DSI implementation files.

Note: It is very important that you take this step to create your own project directory. You might be tempted to just modify the sample project files but we strongly recommend against this, for two reasons:

4. When you install a new release of the SDK, changes you make will be lost.
  5. 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.
2. Open your new directory, and rename the `.vcproj` file located in the `Source` directory. This is the project file for your new ODBC driver, so name it accordingly.
  3. Using a text editor, open the project file (`.vcproj`) and replace every instance of "`QuickstartDSII`" in the source code with the name of your new ODBC driver. Then save and close the file. Do the same for the solution file (`.sln`) as well, making sure that the project file name used is correct.

On Linux and UNIX platforms, you will rename and edit the makefiles to reflect the name for your new ODBC driver.

1. In the `Makefiles` directory, rename the `.mak` file as well as the `.depend` file that is located in the `Makedepend` directory.
2. Using a text editor, open the `Makefile` file located in the `Source` directory. Replace "`Quickstart`" project name in the source code with the name of your new ODBC driver. Then save and close the file.

4. You will need to register your new ODBC driver and create a new ODBC DSN before you will be able to test it. Edit a copy of the appropriate registry (.reg) file located in your source folder. Run this file to register your new driver and then, using the ODBC Data Source Administrator, add a new ODBC DSN that uses it. Point to the new ODBC driver file you will create for registration and for the new ODBC DSN. For now, also point the new ODBC DSN to the Text database folder so the Quickstart code will continue to work until you modify it.

On Linux and UNIX platforms, you will add another ODBC DSN and Driver to your ODBC.INI and ODBCINST.INI files that will point to the new ODBC driver.

5. Build the project to make sure everything compiles. At this point, the new DSII project is identical to the Quickstart Driver example.
6. When you build your new project, you should see the following TODO messages appear in the Output window. You can open the Output window by selecting *Debug-> Windows->Output* from the Main Menu:

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

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

7. As described in steps 6 through 8 of the previous section, use any ODBC-enabled application (e.g. ODBCTest) to test your driver.
8. Similar to step 9 of the previous section, copy and modify the Simba Engine Codebase Example Driver.

By the end of Day One you should have built and tested, then changed, two of the example drivers shipped with SimbaEngine SDK to make sure that your installation worked properly and that your development system is properly set up. Also, you should have created, built and tested a copy of the Quickstart Driver example that you will change to work with your own data store.

## 3.2 Day Two

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

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

The `DSIDriverFactory()` implementation in `Main_Windows.cpp` is the main hook that is called from Simba's ODBC layer to create an instance of your DSI implementation. This method is called as soon as the Driver Manager calls `LoadLibrary()` on your ODBC driver. For the purposes of prototyping, this TODO is purely informational. There is nothing to change here right now, although you may want to add processing at this point for a commercial driver.

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

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

In `QSDriver::SetDriverPropertyValues()` you will set up general properties for your driver. At the very least you will need to change:

- `DSI_DRIVER_NAME` – set this to the name of your driver (the same name you used to replace “QuickstartDSII” in step 3, of Day One).

Depending on the character sets or Unicode encoding used on your data store, you may want to change:

- `DSI_DRIVER_STRING_DATA_ENCODING` – The encoding of char data within the data store. The default value is usually `ENC_UTF8`.
- `DSI_DRIVER_WIDE_STRING_DATA_ENCODING` – The encoding of wide character data within the data store. The default is `ENC_UTF-16LE`.

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

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

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

```
TODO #5: Check Connection Settings.
```

Given a connection string from the ODBC-enabled application, the Simba ODBC layer will parse the connection string into key/value pairs before calling

`QSCONNECTION::UpdateConnectionSettings()` to validate its contents. This method should validate that the entries within `in_connectionSettings` are sufficient to create a connection. If not, you can ask for additional information from the ODBC-enabled application by adding the additional settings to the `out_connectionSettings`.

Should any of the values received be invalid, you should throw an `ErrorException` seeded with `DIAG_INVALID_AUTH_SPEC`. You can also use the utility functions supplied: `VerifyRequiredSetting()` and `VerifyOptionalSetting()`. If there are no further entries required, simply leave `out_connectionSettings` empty.

```
TODO #6: Establish A Connection.
```

Once `QSCONNECTION::UpdateConnectionSettings()` returns `out_connectionSettings` without any required settings (if there are only optional settings, a connection can still occur), the Simba ODBC layer will call `QSCONNECTION::Connect()` passing in *all* the connection settings received from the application. This is where you should authenticate the user against your data store using the information provided within the `in_connectionSettings` parameter.

Should authentication fail, you should throw an `ErrorException` seeded with `DIAG_INVALID_AUTH_SPEC`. You can also use the utility functions supplied: `GetRequiredSetting()` and `GetOptionalSetting()`.

Congratulations! You have now successfully authenticated the user against your data store.

### 3.3 Day Three

Today's goal is to return the data used to pass catalog information back to the ODBC-enabled application. 99.9% of all ODBC-enabled applications require the following ODBC catalog functions:

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

```
TODO #7: Create and return your Metadata Sources.
```

`QSDATAENGINE::MakeNewMetadataTable()` is responsible for creating the sources to be used to return data to the ODBC-enabled application for the various ODBC catalog functions. Each ODBC catalog function is mapped to a unique `DSIMetadataTableId`, which is then

mapped to an underlying `MetadataSource` that you will implement and return. Each `MetadataSource` instance is responsible for three things:

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

## Handling DSI\_TYPE\_INFO\_METADATA

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

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

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

3. For your driver, you may need to change the types returned and the parameters for the types in `QSTypeInfoMetadataSource::PrepareType()`.

## Handling the other MetadataSources

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

1. When called with any other `DSIMetadataTableId`, `QSDatabase::MakeNewMetadataTable()` should return `NULL`. Returning `NULL` will signal SimbaEngine SDK that it should use the metadata helper class returned via `QSDatabase::CreateMetadataHelper()` along with some default `MetadataSources` to create the data source metadata.
2. You will need to change:
  - `QSMetadataHelper::QSMetadataHelper()`  
The example constructor retrieves a list of the tables in the data source. You should modify this method to load the tables defined within your data store.

- `QSMetadataHelper::GetNextTable()`  
In the SimbaEngine Quickstart Driver, this method returns the next table in the data source. You should modify this method to retrieve the next table from your data store.
- The `DSIExtMetadataHelper` class works by retrieving the identifying information for each table and then opening the table via `QSDatabase::OpenTable()`. Once you have implemented `QSTable`, the correct metadata will be returned for all of the tables and columns in your data source.

Congratulations! You can now retrieve type metadata from within your data store. Once you have completed the work for Day Four, you will be able to retrieve the full set of metadata from your data store. You should be able to run `SQLGetTypeInfo()` from within `ODBCTest32.exe` (Unicode) and see the correct metadata returned.

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

## 3.4 Day Four

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

TODO #8: Open A Table.

`QSDatabase::OpenTable()` is the entry point where Simba SQL Engine requests tables involved in the query be opened. You must modify this method to check that the supplied catalog, schema and table names are valid and correspond to a table defined in your data store. If not, you should return null to indicate that the table does not exist.

If the inputs are valid, a new instance of `QSTable` will be returned.

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

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

The next sections describe the changes you must make to `QSTable` for it to work with your data store.

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

- o `QSTable::QSTable()`: The constructor must be modified to take in the catalog, schema and table names and save them in member variables.
- o `QSTable::GetCatalogName()`: Returns `m_catalog`;
- o `QSTable::GetSchemaName()`: Returns `simba_wstring()` (because it does not support schemas);
- o `QSTable::GetTableName()`: Returns `m_table`;
- Return the columns defined for your table.
  - o `QSTable::InitializeColumns()`: This method must be modified so that, for each column defined in the table, you define a `DSIResultSetColumn` in terms of SQL types.

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

```

AutoPtr<DSIResultSetColumns> columns;
Get all the column information from your data store for the table
For Each Defined Column
{
    AutoPtr<DSIColumnMetadata> columnMetadata(
        new DSIColumnMetadata());
    columnMetadata->m_catalogName = m_tableName;
    columnMetadata->m_schemaName = m_schemaName;
    columnMetadata->m_tableName = m_tableName;
    columnMetadata->m_name = //column name
    columnMetadata->m_label = //localized column name
    columnMetadata->m_unnamed = false;
    columnMetadata->m_charOrBinarySize = //the length in bytes
    columnMetadata->m_nullable = DSI_NULLABLE;
    // Change the first parameter of this method to the SQL
    // Type that maps to your data store type.
    SqlTypeMetadata* sqlTypeMetadata =
    SqlTypeMetadataFactory::MakeNewSqlTypeMetadata(SQL_WVARCHAR,
        TDW_BUFFER_OWNED);
    columns->AddColumn(
        new DSIResultSetColumn(
            sqlTypeMetadata,
            columnMetadata.Detach()));
}

m_columns.Attach(columns.Detach());

```

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

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

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

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

- o `QSTable::DoCloseCursor()`

This is a callback method called from Simba SQL Engine to indicate that data retrieval has completed and that you may now do any tasks related to closing the connection to your data store.

Congratulations! You can now retrieve data and see the rest of the metadata from your data store. You should be able to run `SQLTables()` and `SQLColumns()` from within `ODBCTest32.exe` (Unicode) and see the correct metadata returned. You also should be able to execute queries from any ODBC-enabled application (e.g. Microsoft Excel, Microsoft Access, Microsoft SQL Server, and Business Objects Crystal Reports) and see the results returned from your data store.

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

## 3.5 Day Five

Today's goal is to start productizing your driver. Additionally, you can also start localizing your driver error messages. Refer to [SimbaEngine SDK Developer Guide](#) for details on this.

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

All the error messages used within your DSI implementation are stored in a file called `QSMessages.xml`. Rename this file to something appropriate to your data store and update the line associated with the TODO to match.

You should also open the error messages file and change all instances of the following items:

1. The letters “QS” to a two letter abbreviation of your choice
2. The word “Quickstart” to a name relating to your driver

When you are done, you should revisit each exception thrown within your DSI implementation and change the parameters to match as well. This will rebrand your converted SimbaEngine Quickstart Driver for your organization.

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

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

You are now done with all 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 you chose as part of TODO #9.
2. Create a driver configuration dialog. This dialog is presented to the user when they use the ODBC Data Source Administrator to create a new ODBC DSN or configure an existing one. To see this program in operation, open the Control Panel and select Administrative Tools and then select Data Sources (ODBC). Note: Administrative Tools is found under System and Security if your Control Panel is set to view by category.

**IMPORTANT:** If you are using 64-bit Windows with 32-bit applications, you will require special instructions to access the 32-Bit ODBC Data Source Administrator because it is not accessible from the start menu or control panel. Please see Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 27 for details.

The C++ SimbaEngine Quickstart Driver project contains an example ODBC configuration dialog that you may reference for your convenience. You can find the source under the `Setup` folder within the SimbaEngine Quickstart Driver project.

On Linux and UNIX platforms, dialogs are also possible although our Quickstart sample driver for those platforms does not include a sample implementation.

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

The ODBC Data Source Administrator is referenced in several areas of this document. Normally, it is accessed via the Control Panel, under Administrative Tools (which itself is neatly tucked away under System and Security if your Control Panel is set to view by category). It looks the same and is in the same location, whether you are in 32-bit or 64-bit Windows.

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. The ODBC Data Source Administrator that you access through the Control Panel on 64-bit systems is only used by 64-bit applications. 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).

Understanding this, it is very important, when using 64-bit Windows, that you configure the appropriate 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:

Section 2.3.3, “Registry Entries for Drivers and Data Source Names” on page 10 covers the simple case of 32-bit data sources on 32-bit Windows. The sections below provide details regarding 64-bit Windows.

### 32-Bit Drivers on 64-Bit Windows

The 32-bit applications and drivers see a subsection 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 three string values to define the location of the **Driver**, the database (DBF) that it will use and a **Description** to help you clearly identify each registry key. The three keys that are relevant to the the C++ examples discussed in this document are:

- **QuickstartDSII** which must include the following string values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartDSII.dll*
  - **DBF:** *[INSTALL\_DIRECTORY]\Examples\Databases\Text*
  - **Description:** Sample 32-bit SimbaEngine Quickstart DSII
- **CodebaseDSII** which must include the following string values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\CodebaseDSII.dll*
  - **DBF:** *[INSTALL\_DIRECTORY]\Examples\Databases\Codebase*
  - **Description:** Sample 32-bit SimbaEngine Codebase DSII
- **UltraLightDSII** which must include the following string values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightDSII.dll*
  - **Description:** Sample 32-bit SimbaEngine Codebase DSII

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

- **ODBC Data Sources** which must include the following string values:
  - **QuickstartDSII:** *QuickstartDSIIDriver*
  - **CodebaseDSII:** *CodebaseDSIIDriver*
  - **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:

- **QuickstartDSIIDriver** which includes the following key names and values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartDSII.dll*
  - **Setup:** *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartDSII.dll*
  - **Description:** Sample 32-bit SimbaEngine Quickstart DSII
- **CodebaseDSIIDriver** which includes the following key names and values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\CodebaseDSII.dll*
  - **Setup:** *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\CodebaseDSII.dll*
  - **Description:** Sample 32-bit SimbaEngine Codebase DSII
- **UltraLightDSIIDriver** which includes the following key names and values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightDSII.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:
  - **QuickstartDSIIDriver:** *Installed*
  - **CodebaseDSIIDriver:** *Installed*
  - **UltraLightDSIIDriver:** *Installed*

## 64-Bit Drivers on 64-Bit Windows

On a 64-bit machine, only 64-bit applications can see the 64-bit registry and the 64-bit ODBC drivers and data sources contained in it. The SimbaEngine SDK installer itself is a 32-bit application, so it can only pre-create 32-bit data sources whether it is on a 32-bit or a 64-bit Windows machine. If you are using 64-bit Windows, you will not be able to use the example drivers “out of the box” with 64-bit applications. You will first need to add the registry entries necessary for the sample drivers.

In the *[INSTALL\_DIRECTORY]\Setup* folder, there is a registry file *SEN9Setup64Bit.reg*. Edit this file to replace *[INSTALL\_DIRECTORY]* with the path to where the SDK was installed. Take note that you must enter double backslashes in the folder path or the entries will not be created. Run this file to update your Windows registry.

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 location of the **Driver**, the database (**DBF**) that it will use and a **Description** to help you clearly identify each registry key. The three keys that are relevant to the C++ examples discussed in this document are:

- **QuickstartDSII** which must include the following string values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\x64\Release\QuickstartDSII.dll*
  - **DBF:** *[INSTALL\_DIRECTORY]\Examples\Databases\Text*
  - **Description:** Sample 64-bit SimbaEngine Quickstart DSII
- **CodebaseDSII** which must include the following string values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\x64\Release\CodebaseDSII.dll*
  - **DBF:** *[INSTALL\_DIRECTORY]\Examples\Databases\Codebase*
  - **Description:** Sample 64-bit SimbaEngine Codebase DSII
- **UltraLightDSII** which must include the following string values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\x64\Release\UltraLightDSII.dll*
  - **Description:** Sample 64-bit SimbaEngine UltraLight DSII

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

- **ODBC Data Sources** which must include the following string values:
  - **QuickstartDSII:** *QuickstartDSIIDriver*
  - **CodebaseDSII:** *CodebaseDSIIDriver*
  - **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:

- **QuickstartDSIIDriver** which includes the following key names and values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\x64\Release\QuickstartDSII.dll*
  - **Setup:** *[INSTALL\_DIRECTORY]\Examples\Bin\x64\Release\QuickstartDSII.dll*
  - **Description:** Sample 64-bit SimbaEngine Quickstart DSII
- **CodebaseDSIIDriver** which includes the following key names and values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\x64\Release\CodebaseDSII.dll*
  - **Setup:** *[INSTALL\_DIRECTORY]\Examples\Bin\x64\Release\CodebaseDSII.dll*

- **Description:** Sample 64-bit SimbaEngine Codebase DSII
- **UltraLightDSIIDriver** which includes the following key names and values:
  - **Driver:** *[INSTALL\_DIRECTORY]\Examples\Bin\x64\Release\UltraLightDSII.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:
  - **QuickstartDSIIDriver:** *Installed*
  - **CodebaseDSIIDriver:** *Installed*
  - **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. `QSTable::GetData()`

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

In addition, both methods accept the following three parameters:

1. `in_data`

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

The type of this parameter is governed by the metadata for the column that is returned by the class. Thus, if you set the SQL Type of column 1 in `QSTable::InitializeColumns()` to `SQL_INTEGER`, then when `QSTable::GetData()` is called for column 1, you will be passed a `SQLData` that wraps an `int` 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 `QSTypeUtilities.h` for an example on how to handle character or binary data.

2. `in_offset`

Some 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 type. For character or binary data, copying data over this amount can result in a data truncation warning, or worse, a heap-violation.

## SqlData types

SqlData objects essentially 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, you can use `GetMetadata()->GetSqlType()` to see what the associated SQL\_\* type is.

For information how SQL types map to C++ types, see Appendix H in the Developer's Guide.

### Fixed Length Types:

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

SQL\_BIT

SQL\_DECIMAL

SQL\_DOUBLE

SQL\_FLOAT

SQL\_NUMERIC

SQL\_REAL

SQL\_SBIGINT

SQL\_SINTEGER

SQL\_SSMALLINT

SQL\_STINYINT

SQL\_TYPE\_DATE

SQL\_TYPE\_TIME

SQL\_TYPE\_TIMESTAMP

SQL\_UBIGINT

SQL\_UINTEGER

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

Note: 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");
        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);
    }
}
```

## Data Conversion in Practice

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

For your data source, if you're already getting data of the correct type (i.e. integers, etc.) then ideally you should skip the conversion to strings so you can achieve better performance. What you need is to be aware of which data types map to which SQL Types (and how to represent them in the expected format). Then you can set the buffer in an appropriate manner.

## NULL Values

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

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

## Appendix D: How to Add Schema Support

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

### Handling DSI\_SCHEMAONLY\_METADATA

1. `QSCConnection::SetConnectionPropertyValues()` currently disables schema support via `DSIPropertyUtilities::SetSchemaSupport()`. Change this value to `true` to enable schema support.
2. You will also need to change:
  - a. `QSMetadataHelper::GetNextTable()`  
In the SimbaEngine Quickstart Driver a blank schema is returned as schema support is not enabled by default. The schema will need to be returned in the Identifier to allow SimbaEngine SDK to open the correct table.
  - b. `QSDataEngine::OpenTable()`  
Modify this method to verify the given schema and return the correct table for the given catalog, schema, and table name.
  - c. `QSTable::GetSchemaName()`  
Modify this method to return the schema the table belongs to.

## Appendix E: C++ Server Configuration

To establish a connection, the connection settings for the driver are normally retrieved directly from the ODBC DSN. When the driver is a server, however, 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 on page 28 for details.

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

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

To set the Quickstart sample driver up as a server:

1. Build the *Quickstart* solution using a server configuration (i.e. *Debug\_Server* or *Release\_Server*). This should build the server executable.
2. Create the registry key `HKEY_LOCAL_MACHINE/SOFTWARE/SIMBA/QUICKSTART/SERVER`, and add the following string value:
  - a. `DBF=[INSTALL_DIRECTORY]\Examples\Databases\Text`
3. The rest of the server settings are located under sub-nodes of `HKEY_LOCAL_MACHINE/SOFTWARE/SIMBA/QUICKSTART/SERVER`. For full list of possible server configuration parameters, please see the SimbaClientServer User Guide.

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

1. Build Quickstart using the debug (or release) server configuration:
 

```
BUILDSERVER=exe make -f Quickstart.mak debug
```
2. Add the DBF value to the [Server] section of the `.simbaserver.quickstart.ini` file:
 

```
DBF=[INSTALL_DIRECTORY]/Examples/Databases/Text
```
3. Configure the server as required in the other sections of the `.simbaserver.quickstart.ini` file.

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