



SimbaEngine SDK 9.0

Build a Java 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 8th Avenue
Vancouver, BC Canada
V5Z 1E5

Tel. +1.604.633.0008
Fax. +1.604.633.0004

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.
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 Folders 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.....	12
3.1	Day One.....	13
3.2	Day Two.....	20
3.3	Day Three.....	22
3.4	Day Four.....	24
3.5	Day Five.....	27
Appendix A:	ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit.....	29
Appendix B:	Windows Registry 32-Bit vs. 64-Bit.....	30
Appendix C:	Data Retrieval.....	33
Appendix D:	How to Add Schema Support.....	35
Appendix E:	Java Server Configuration.....	36

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 Java as your development environment.

There are five sample drivers provided with the SimbaEngine SDK:

1. The activities in this guide use the JavaQuickstart sample driver (which accesses the most basic type of text-based data source) as a base from which you will work.
2. The Quickstart 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*.
3. The Codebase sample driver provides implemented C++ 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.
4. 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*.
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 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 *[INSTALL_DIRECTORY]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 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>/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:** 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:** (*this document*) 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, 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.

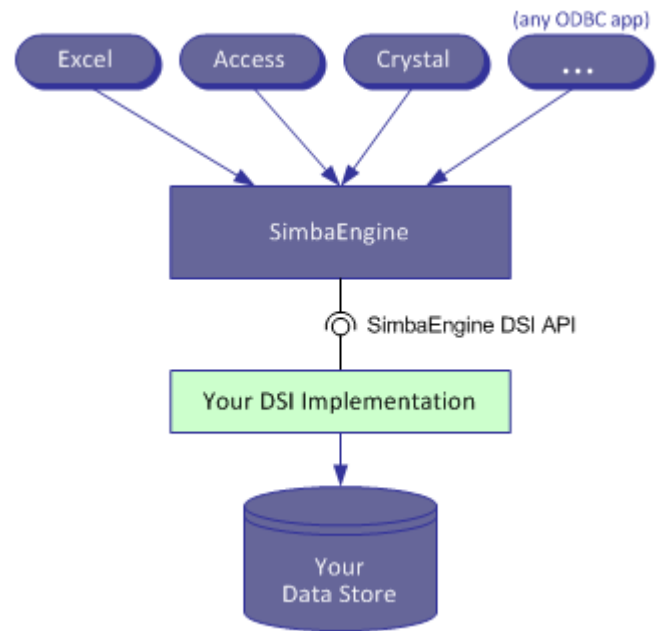


Figure 1: High level view of SimbaEngine

The sample projects provided with the SDK are functioning DSI Implementations that you can copy and modify for your own purposes. The JavaQuickstart example that is the primary subject of this document connects to a simple text file. Following the steps in this guide, you will change the JavaQuickstart 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 DSII 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 directly through to your database. The prototype you will build by following this document accesses a non-SQL data source and thus uses SQLEngine.

On Linux 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 JavaQuickstart driver code we have highlighted the areas you need to change by adding comments prefixed with “TODO” so you can find them, along with a short explanatory message. There are areas in the code highlighted with `TODOs` and this document will walk you through each of them.

Of the areas of the code that you need to modify, most 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 error file and log files. In other words, these are not complicated tasks.

The other 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 JavaQuickstart 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. These are called `QUtilities` in the Simba JavaQuickstart Driver. If your resultant DSI implementation design looks like this, you are on the right track.

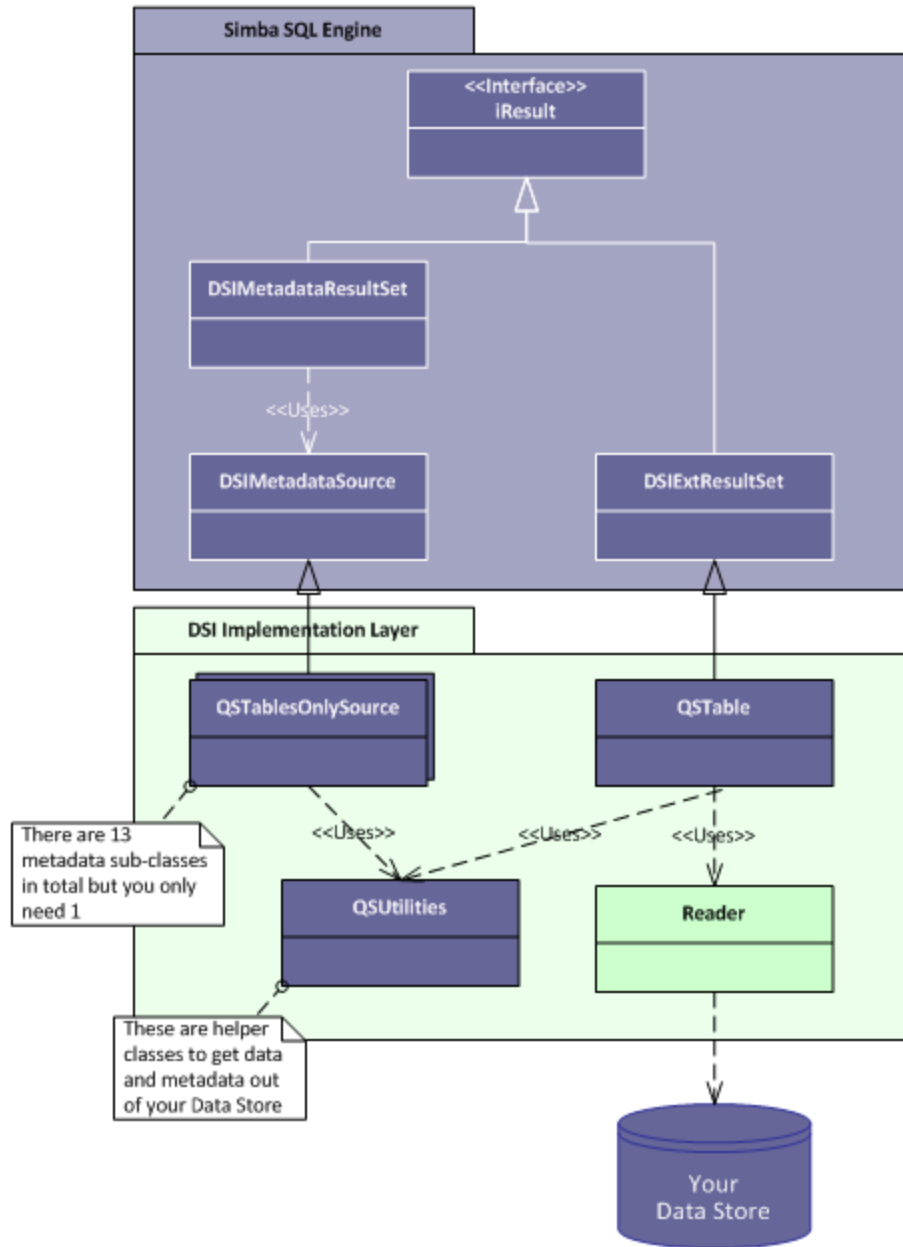


Figure 2: Design pattern for a DSI implementation.

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>/SimbaEngineSDK/9.0

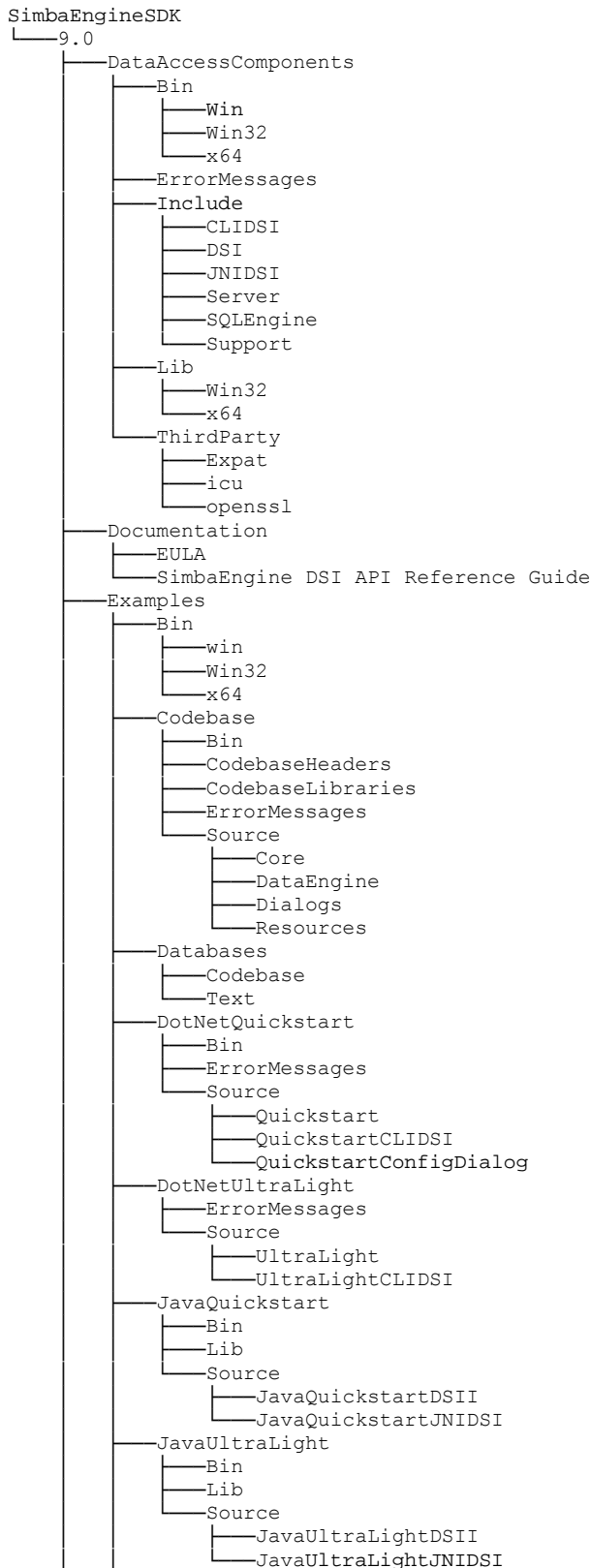
Note: The exact install directory tree structure for Linux/Unix/MacOSX is slightly different than the Windows tree shown below, but the general layout is similar.

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\



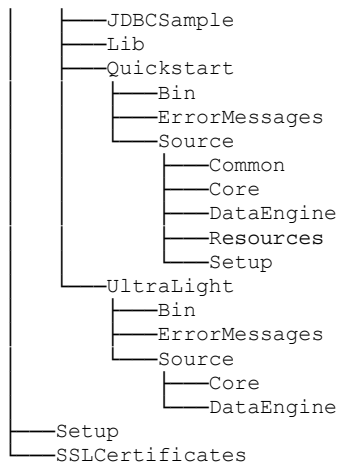


Figure 3: The structure of the folders installed by SimbaEngine SDK.

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 30 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 two that are relevant to the Java examples discussed in this document:

- **JavaQuickstartDSII** which includes the following key names and values:
 - **Driver:** *[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartJNIDSII.dll*
 - **DBF:** *[INSTALL_DIRECTORY]\Examples\Databases\Text*
 - **Description:** Sample 32-bit SimbaEngine JavaQuickstart DSII
- **JavaUltraLightDSII** which includes the following key names and values:
 - **Driver:** *[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightJNIDSII.dll*
 - **DBF:** *[INSTALL_DIRECTORY]\Examples\Databases\Text*
 - **Description:** Sample 32-bit SimbaEngine JavaUltraLight 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:
 - **JavaQuickstartDSII:** *JavaQuickstartDSIIDriver*
 - **JavaUltraLightDSII:** *JavaUltraLightDSIIDriver*

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 two that are relevant to the Java examples discussed in this document:

- **JavaQuickstartDSIIDriver** which includes the following key names and values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartJNIDSI.dll`
 - **Description:** Sample 32-bit SimbaEngine Java JavaQuickstart DSII
- **JavaUltraLightDSIIDriver** which includes the following key names and values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightJNIDSI.dll`
 - **Description:** Sample 32-bit SimbaEngine Java JavaUltraLight 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:
 - **JavaQuickstartDSIIDriver:** *Installed*
 - **JavaUltraLightDSIIDriver:** *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 13), 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 the supplied example project to create new entries in `ODBCINST.INI` and 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 29 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 JavaQuickstart 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"> • Install the SimbaEngine SDK 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 driver using an ODBC-enabled application. • Set up a new project directory where you will begin to modify the sample driver as the starting point for your new driver.
Two	<ul style="list-style-type: none"> • Set the driver properties to give your driver a name. • Set the driver-wide and connection-wide logging level. • Modify the connection setting validation method to match the connection string needed for your data store. • Modify the connection authentication method to match the settings needed for your data store.
Three	<ul style="list-style-type: none"> • Modify the method used to create and return your Metadata Sources, which will support ODBC catalog functions. • Modify the method used to support SQLGetTypeInfo ODBC catalog function with the data types supported by your data store. • 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.
Four	<ul style="list-style-type: none"> • Modify the method used to open a table defined within your data store. • Modify the method used to retrieve the column information for a table in your data store. • Modify the methods used to navigate through and retrieve data from your data store.
Five	<ul style="list-style-type: none"> • Register your error message file. • Modify the parameters of the exceptions thrown by your driver to match as well. • Rename the packages, files and classes to use a name and abbreviation appropriate for your driver.

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, built and tested your first ODBC driver.

Initial Set Up

Start by installing the SDK and running the example drivers:

1. Install the SimbaEngine SDK using the setup executable for your version of Visual Studio – run this program and follow the installer's instruction. The default locations and resulting folders are shown in Figure 3, on page 9. **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 Examples folder. These instructions assume you will be working with the 32-bit version of the Visual Studio 2008 `JavaQuickstart` Driver example as well as an Eclipse integrated development environment (IDE) on a 32-bit Windows operating system with a version 1.5 or higher 32-bit JDK. Ensure that you have the environment variable `JAVA_HOME` pointing to the root

of your 32-bit JDK and that PATH includes the path to the jvm.dll. (Note that for 64-bit drivers, a 64-bit JDK will be required with JAVA_HOME and PATH configured accordingly.)

2. Open the `QuickstartJNIDSI_net2008.sln` project file, located in `[INSTALL_DIRECTORY]\Examples\JavaQuickstart\Source\JavaQuickstartJNIDSI`. Note that this solution contains the `QuickstartJNIDSI`, which is the driver's native component (C++). Every Java ODBC driver built using the SimbaEngine SDK will have two components, one native and one Java. The native component (in this case `QuickstartJNIDSI`) has only a couple basic functions: to find or create an instance of the Java Virtual Machine (JVM) and to provide the Java driver implementation name to the bridge between C++ and Java (`SimbaJNIDSI`). The Java component (in this case `JavaQuickstart`) is the DSII.
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 native driver. This will build the debug version of the driver and place it in the location `[INSTALL_DIRECTORY]\Examples\JavaQuickstart\Bin\Win32\Debug`

On Linux 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 `JavaQuickstart` 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\JavaQuickstart\Makefiles` directory.
2. Set the `SIMBAENGINE_DIR` and `JNIDSI_DIR` environment variables:


```
export
SIMBAENGINE_DIR=/<theUntarDirectory>/SimbaEngineSDK/9.0/DataAccessCo
mponents
export JNIDSI_DIR=$SIMBAENGINE_DIR
```
3. Ensure that you have the environment variable `JAVA_HOME` pointing to the root of your 32-bit JDK and that your `JAVA_HOME`'s `bin` directory is on your `PATH`.
4. 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 QuickstartJNIDSI.mak debug
```

4. Use Eclipse to import the `JavaQuickstart` project as an existing project into your workspace. The DSII project files are located under `[INSTALL_DIRECTORY]\Examples\JavaQuickstart\Source\JavaQuickstartDSII`. From the Main Menu select `Project->Properties->Java Build Path->Libraries`. Select the `SIMBAENGINE_DIR` entry, `Edit, Variable, New` and create a new classpath variable named `SIMBAENGINE_DIR` with the Path pointing to the `DataAccessComponents` folder of your

installed SimbaEngine SDK. Select OK and do a full rebuild of the workspace when prompted. When you see that there are no errors, you are ready to build the driver.

5. From the Main Menu select Run->External Tools->JavaQuickstart
JavaQuickstartBuilder.xml. This will build the Java driver using ANT and place it in the location `[INSTALL_DIRECTORY]\Examples\JavaQuickstart\Lib`.
6. 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 `SetupMyJavaQuickstartDSII-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 30).

For this Day One example, you need to replace or update several items. Take note that you must enter double backslashes in the folder paths or the entries will not be created:

- 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).
 - Make sure that the DBF points to the correct location. This is where the Tabbed Unicode database files are that the JavaQuickstart sample driver can read.
 - Make sure that the `-Djava.class.path` path value for `JavaQuickstart.jar` points to the correct location
 - (Optional) Give your new DSN a name by replacing `MyJavaQuickstartDSII`. You can also change or remove the description if you want to.
 - Run this file and it will update your Windows registry to register the ODBC driver and your new ODBC DSN.
7. 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 29 for details.

In the System DSN tab, check that your DSN has been registered. Note that you will not be able to add or configure the JavaQuickstart driver through this interface because the sample does not include a driver configuration dialog. (This can be added in later: for more information please see the description at the end of On Linux platforms, lists of catalogs, schemas, tables and types are available using the `qualifiers`, `owners`, `tables` and `types` commands in the `iodbctest` utility.

Day Five.)

On Linux and UNIX platforms, to configure the Data Sources for your driver you will need to edit the ODBC.INI, ODBCINST.INI, and SIMBA.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]
MyJavaQuickstartDSII=MyJavaQuickstartDSIIDriver

[MyJavaQuickstartDSII]
Description=Sample 32-bit SimbaEngine JavaQuickstart DSII
DBF=<theUntarDirectory>/SimbaEngineSDK/9.0/Examples/Databases/Text/
Driver=<theUntarDirectory>/SimbaEngineSDK/9.0/Examples/JavaQuickstart/
Bin/Linux_x86/libQuickstartJNIDSI_debug.so
```

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

```
[ODBC Drivers]
MyJavaQuickstartDSIIDriver=Installed

[MyJavaQuickstartDSIIDriver]
Driver=<theUntarDirectory>/SimbaEngineSDK/9.0/Examples/JavaQuickstart/
Bin/Linux_x86/libQuickstartJNIDSI_debug.so
```

3. Edit the .simba.ini file and add:

```
[Driver]
JNIConfig=-
Djava.class.path=<theUntarDirectory>/SimbaEngineSDK/9.0/Examples/JavaQuickstart/
Lib/JavaQuickstart.jar|-Xdebug|-
Xrunjdpw:transport=dt_socket,address=localhost:8000,suspend=n,server=y
```

4. 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
```

8. You should now be able to place breakpoints anywhere in the SimbaEngine JavaQuickstart DSI implementation. A good breakpoint to start with is the QSDriver constructor. This code runs as soon as the Java driver is loaded.
9. From the Main Menu select Run->Debug Configurations... -> Remote Java Applications and press the New button. Enter an appropriate name for the configuration, select the JavaQuickstart project, and edit the port to match the listen port specified in the

JNIConfig entry in the registry (located under *HKEY_LOCAL_MACHINE/SOFTWARE/SIMBA/DRIVER*). Also check that the debug configuration is set to `suspend=y` in the JNIConfig (i.e. `...JavaQuickstart.jar|-Xdebug|-Xrunjdwp:transport=dt_socket,address=localhost:8000,suspend=y,server=y`). Leave the Debug Configurations window open.

10. Open any ODBC-enabled application (e.g. ODBC Test) and execute an ODBC operation within the application. The application should wait for you to attach the Eclipse debugger. Switch to Eclipse and select Debug in the Debug Configuration window that you left open in the previous step. You should hit the breakpoint you created.

On Linux 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 test utility such as `iodbctest`.

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

```
export
LD_LIBRARY_PATH=$SIMBAENGINE_DIR/ThirdParty/icu/Linux_x86/lib:$JAVA_HOME/jre/lib/i386/client:$LD_LIBRARY_PATH
```

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

```
iodbctest DSN=MyJavaQuickstartDSII
```

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 `JavaQuickstart` directory and paste it to the same location. This will create a new directory called `"JavaQuickstart - 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 `[INSTALL_DIRECTORY]\Examples\JavaQuickstart\Source\JavaQuickstartJNIDSJ` sub-directory. This is the project file for your new ODBC driver, so name it accordingly. Using a text editor, open the project file (`.vcproj`) and replace every instance of “QuickstartJNIDSJ” in the source code with the name of your new ODBC driver followed with JNIDSJ. Then save and close the file.

On Linux 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 the “QuickstartJNIDSJ” project name in the source code with the name of your new ODBC driver. Then save and close the file.

3. Similarly, rename the `.xml` file located in the `[INSTALL_DIRECTORY]\Examples\JavaQuickstart\Source\JavaQuickstartDSJ` sub-directory. This is the ANT builder file (`.xml`) for your new ODBC driver. Using a text editor, open the ANT builder (`.xml`) and replace every instance of “JavaQuickstart” and “JavaQS” in the source code with the name of your new ODBC driver. Also, you will likely wish to either remove or replace the copyright information for the “doc” target. 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 DSN. 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 JavaQuickstart code will continue to work until you modify it.

On Linux 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. You should also ensure that the `JNIConfig` in your `SIMBA.INI` file is also updated.

5. Build the project to make sure everything compiles. At this point, the new `DSJ` project is identical to the JavaQuickstart Driver example.
6. Searching your C++ solution for ‘TODO’ will find the following comments marking locations at which changes in your driver need to be made:

```
TODO #1: Find or create the Java Virtual Machine.      (QuickstartJNIDSJ.cpp)
TODO #2: Update full Java driver name.                (QuickstartJNIDSJ.cpp)
```

7. Searching your Java workspace for ‘TODO’ will find the following comments marking locations at which changes in your driver need to be made:

TODO #1: Set the driver properties.	(QSDriver.java)
TODO #2: Set the driver-wide logging details.	(QSDriver.java)
TODO #3: Set the connection-wide logging details	(QSConnection.java)
TODO #4: Check Connection Settings.	(QSConnection.java)
TODO #5: Establish A Connection.	(QSConnection.java)
TODO #6: Update configuration file.	(QSConnection.java)
TODO #7: Create and return your Metadata Sources.	(QSDataEngine.java)
TODO #8: Open A Table.	(QSDataEngine.java)
TODO #9: Assign a unique component ID.	(QSDriver.java)
TODO #10: Update Messages properties file.	(QSDriver.java)
TODO #11: Create an ExceptionBuilder.	(QSDriver.java)
TODO #12: Register the Quickstart messages.	(QSDriver.java)

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

8. As described in the previous section, use any ODBC-enabled application (e.g. ODBC Test) to test your driver.

By the end of this day you should have built and tested, unchanged, the example driver 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 JavaQuickstart 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 the C++ TODO item 1 and the Java TODO items 1 to 6.

TODO #1: Find or create the Java Virtual Machine. (QuickstartJNIDSI.cpp)

The `JvmFactory()` implementation in `QuickstartJNIDSI.cpp` in the *QuickstartJNIDSI* project is the first hook that is called from Simba's JNIDSI layer to find or create an instance of the Java VM during initialization of the bridge. This method is called soon after the Driver Manager calls `LoadLibrary()` on your ODBC driver. After that, the C++ to Java proxies are initialized and an instance of your DSI implementation is created when the name of the Java Driver is retrieved from `GetFullJavaDriverName()` (but more on that later). 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.

TODO #1: Set the driver properties. (QSDriver.java)

Moving on to the *JavaQuickstart* project, in `QSDriver`'s `setDefaultProperties()` 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 “JavaQuickstart” in step 3, of Day One).

TODO #2: Set the driver-wide logging details. (QSDriver.java)

TODO #3: Set the connection-wide logging details. (QSConnection.java)

By default, the SimbaEngine JavaQuickstart 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 #4: Check Connection Settings. (QSConnection.java)

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`'s `updateConnectionSettings()` to validate its contents. This method should validate that the entries within the `requestMap` are sufficient to create a connection. If not, you can ask for additional information from the ODBC-enabled application by specifying the additional settings in the return value.

Should any of the values received be invalid, you should throw a `BadAuthException`. For your convenience, you can also use the utility functions supplied:

`verifyRequiredSetting()` and `verifyOptionalSetting()`. If there are no further entries required, simply leave the `responseMap` empty.

```
TODO #5: Establish A Connection. (QSConnection.java)
```

Once `QSConnection`'s `updateConnectionSettings()` returns a `responseMap` without any required settings (if there are only optional settings, a connection can still occur), the Simba ODBC layer will call `QSConnection`'s `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 `requestMap` parameter.

Should authentication fail, you should throw a `BadAuthException`. You can also use the utility functions supplied: `getRequiredSetting()` and `getOptionalSetting()`.

```
TODO #6: Update configuration file. (QSConnection.java)
```

The purpose of this configuration file is to enable server-specific behavior during runtime. When the driver is configured to be a server, then the connection settings in `QSConnection`'s `updateConnectionSettings()` and `connect()` need to be augmented. This file only needs to exist if driver is set to be a server.

For the purposes of prototyping, this TODO is mostly informational. If you know that you will use the client/server configuration for accessing your data source, rename the configuration file to use an appropriate name. Otherwise there is nothing to change here right now, though you may wish to revisit this later. For more information on setting up a Java driver as a server, please refer to Appendix E: Java Server Configuration.

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 return catalog information 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.java`)

`QSDataEngine`'s `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 `MetadataSourceId`, which is then mapped to an underlying `IMetadataSource` that you will implement and return. Each `IMetadataSource` 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: `moveToNextRow()`
3. Retrieving data: `getMetadata()` (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 `TYPE_INFO`, `QSDataEngine`'s `makeNewMetadataTable()` will return an instance of `QSTypeInfoMetadataSource()`.
2. The SimbaEngine JavaQuickstart 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`'s `initializeDataTypes()`.

Handling the other MetadataSources

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

1. When called with any other `MetadataSourceId`, `QSDatabaseEngine`'s `makeNewMetadataTable()` should return `NULL`. Returning `NULL` will signal SimbaEngine SDK that it should use the metadata helper class returned via `QSDatabaseEngine`'s `createMetadataHelper()` along with some default `IMetadataSources` to create the data source metadata.
2. You will need to change:
 - `QSMetadataHelper`'s `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`'s `getNextTable()`
In the SimbaEngine JavaQuickstart 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.
 - `QSMetadataHelper`'s `getNextProcedure()`
In the SimbaEngine JavaQuickstart Driver, this method returns the next procedure in the data source. If procedures are supported, you should modify this method to return to retrieve the next procedure from your data store, or just return `false`.
 - The `DSIExtMetadataHelper` class works by retrieving the identifying information for each table and then opening the table via `QSDatabaseEngine`'s `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 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. (QSDataEngine.java)
```

`QSDataEngine`'s `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 `DSIExtResultSet`, 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 SimbaEngine JavaQuickstart Driver for Tabbed Unicode Files. The SimbaEngine JavaQuickstart 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:
 - `QSTable`'s `QSTable()`: The constructor must be modified to take in the catalog, schema and table names and save them in member variables.
 - `QSTable`'s `getCatalogName()`: Returns `Quickstart.CATALOG` (because it has only a single catalog);
 - `QSTable`'s `getSchemaName()`: Returns `null` (because it does not support schemas);
 - `QSTable`'s `getTableName()`: Returns `m_tableName`;
- Return the columns defined for your table.
 - `QSTable`'s `initializeColumns()`: This method must be modified so that, for each column defined in the table, you define the `ColumnMetadata` in terms of SQL types.

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

```
Get all the column information from your data store for the table
For Each Defined Column
{
    // Change the parameter of this method to the SQL Type that
```

```

// maps to your data store type.
TypeMetadata typeMetadata =
TypeMetadata.createTypeMetadata(Types.VARCHAR);

// Depending on SQL type, set different properties:
if (character type)
{
    typeMetadata.setIntervalPrecision(m_settings.m_maxColumnSize);
}
else if (exact numeric type)
{
    typeMetadata.setScale( scale );
}

ColumnMetadata columnMetadata = new ColumnMetadata(typeMetadata);
columnMetadata.setCatalogName(m_catalogName);
columnMetadata.setSchemaName(m_schemaName);
columnMetadata.setTableName(m_tableName);
columnMetadata.setName("column name");
columnMetadata.setLabel("localized column name");
columnMetadata.setNullable(Nullable.NULLABLE);

if ( character type )
{
    columnMetadata.setColumnLength(m_settings.m_maxColumnSize);
}

m_columns.add(columnMetadata);
}

```

- Data Retrieval

- o QSTable's moveToNextRow()
- o QSTable's getData()

These 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 SimbaEngine JavaQuickstart 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 `column` index 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's closeCursor()

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, Business Objects Crystal Reports) and see the results returned from your data store.

On Linux 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 the SimbaEngine SDK Developer Guide for details on this.

```
TODO #9: Assign a unique component ID. (QSDriver.java)
```

For the purpose of prototyping, this TODO is purely informational. The default component ID is usually sufficient for most drivers. The component ID is used to identify the component from which an error has been generated so that the correct component name can be included in the error message.

```
TODO #10: Update Messages properties file. (QSDriver.java)
```

All the error messages used within your DSI implementation are stored in a file called `messages.properties`. You should revisit each exception thrown within your DSI implementation and change the parameters to match as well.

```
TODO #11: Create an ExceptionBuilder. (QSDriver.java)
```

For the purpose of prototyping, this TODO is purely informational. The `ExceptionBuilder` that is created by default should be sufficient for the vast majority of drivers. However, if you wish to extend the `ExceptionBuilder` class, this is the location where you would instantiate your class.

```
TODO #12: Register the Quickstart messages. (QSDriver.java)
```

For the purpose of prototyping, this TODO is mostly informational. By default, the driver's `messages.properties` file resides in the same package as the `QSDriver` class. You can modify the code to look in a different package location for the messages file or to customize the name of the file.

All error messages returned by the driver begin with the component name. Simply change the "Quickstart" to a name relating to your driver. This will rebrand your converted SimbaEngine JavaQuickstart Driver for your organization.

```
TODO #2: Update full Java driver name. (QuickstartJNIDSI.cpp)
```

First rename all packages, files, and classes by changing all instances of the following items:

- The word "Quickstart" to the name you chose as a part of TODO #9.
- The letters "QS" to a two letter abbreviation of your choice.

Now you can update the full Java driver name for this TODO to match the path and name of your driver. Simply replace the package name “quickstart” with your re-branded name as well as the “QS” for the Driver name with your two letter abbreviation.

You are now done with all the TODO’s in the project. However, there is still one final step before you have a fully functioning driver:

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 the *Quickstart* driver (note: this is not the JavaQuickstart driver) from 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 29 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 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 two that are relevant to the Java examples discussed in this document:

- **JavaQuickstartDSII** which must include the following string values:
 - **Driver:** *[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartJNIDSI.dll*
 - **DBF:** *[INSTALL_DIRECTORY]\Examples\Databases\Text*
 - **Description:** Sample 32-bit SimbaEngine JavaQuickstart DSII
- **JavaUltraLightDSII** which must include the following string values:
 - **Driver:** *[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightJNIDSI.dll*
 - **Description:** Sample 32-bit SimbaEngine JavaUltraLight 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:
 - **JavaQuickstartDSII:** *JavaQuickstartDSIIDriver*
 - **JavaUltraLightDSII:** *JavaUltraLightDSIIDriver*

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 two that are relevant to the Java examples discussed in this document:

- **JavaQuickstartDSIIDriver** which includes the following key names and values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\QuickstartJNIDSI.dll`
 - **Description:** Sample 32-bit SimbaEngine JavaQuickstart DSII
 - **Setup:** (Not included in this sample.)
- **JavaUltraLightDSIIDriver** which includes the following key names and values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightJNIDSI.dll`
 - **Description:** Sample 32-bit SimbaEngine JavaUltraLight DSII
 - **Setup:** (Not included in this sample.)

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:
 - **JavaQuickstartDSIIDriver:** *Installed*
 - **JavaUltraLightDSIIDriver:** *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 two that are relevant to the Java examples discussed in this document:

- **JavaQuickstartDSII** which must include the following string values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\x64\Release\QuickstartJNIDSI.dll`
 - **DBF:** `[INSTALL_DIRECTORY]\Examples\Databases\Text`
 - **Description:** Sample 64-bit SimbaEngine JavaQuickstart DSII
- **JavaUltraLightDSII** which must include the following string values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\x64\Release\UltraLightJNIDSI.dll`
 - **Description:** Sample 64-bit SimbaEngine JavaUltraLight 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:
 - **JavaQuickstartDSII:** `JavaQuickstartDSIIDriver`
 - **JavaUltraLightDSII:** `JavaUltraLightDSIIDriver`

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 two that are relevant to the Java examples discussed in this document:

- **JavaQuickstartDSIIDriver** which includes the following key names and values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\x64\Release\QuickstartJNIDSI.dll`
 - **Description:** Sample 64-bit SimbaEngine JavaQuickstart DSII
 - **Setup:** (Not included in this sample.)
- **JavaUltraLightDSIIDriver** which includes the following key names and values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\x64\Release\UltraLightJNIDSI.dll`
 - **Description:** Sample 64-bit SimbaEngine JavaUltraLight DSII
 - **Setup:** (Not included in this sample.)

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:
 - **JavaQuickstartDSIIDriver:** `Installed`
 - **JavaUltraLightDSIIDriver:** `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 `IMetadataSource` implementation of `getMetadata()`
2. `QSTable`'s `getData()`

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

In addition, both methods accept the following three parameters:

3. `data`

The `DataWrapper` into which you must copy your cell's value. This class is a wrapper around an Object managed by the Simba SQL Engine. You simply call its `set<DataType>()` and `get<DataType>()` methods to store and access the data according to the `java.sql.Type`. The data you set must be represented as the Object or primitive data type that is accepted by the set methods for that `java.sql.Type`. If your data is not stored as the appropriate type, 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 create the `TypeMetadata` of column 1 in `QSTable`'s `initializeColumns()` as `Types.INTEGER`, then when `QSTable`'s `getData()` is called for column 1, you will be passed a `DataWrapper` that wraps a `Long` data type. For `IMetadataSource`, the type is associated with the column tag (see `MetadataSourceColumnTag`).

It is important to note that while Java does not natively support unsigned integer-types (i.e. the types represented by `TINYINT`, `SMALLINT`, `INTEGER`), the Simba SQL Engine allows for unsigned data types to be retrieved through the C++ to Java bridge. By up-casting to a larger signed type for each of the integer-types, unsigned values can be stored until they are retrieved and converted to the correct unsigned SQL Type at the C++ end of the bridge. By default, the `TypeMetadata` for the column is set to treat the integer-types as signed. To enable unsigned data, you will need to call `TypeMetadata`'s `setSigned(true)` when creating the `ColumnMetadata` for the column in `QSTable`'s `initializeColumns()`.

4. `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.

5. `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.

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. `QSConnection`'s `setDefaultProperties()` currently disables schema support via `PropertyUtilities`'s `setSchemaSupport()`. Change this value to `true` to enable schema support.
2. You will also need to change:
 - a. `QSMetadataHelper`'s `getNextTable()`
In the SimbaEngine JavaQuickstart driver a null 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`'s `openTable()`
Modify this method to verify the given schema and return the correct table for the given catalog, schema, and table name.
 - c. `QSTable`'s `getSchemaName()`
Modify this method to return the schema the table belongs to.
3. If procedures are supported, you will also need to change:
 - d. `QSMetadataHelper`'s `getNextProcedure()`
In the SimbaEngine JavaQuickstart driver a null 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 procedure.
 - e. `QSDataEngine`'s `openProcedure()`
Modify this method to verify the given schema and return the correct procedure for the given catalog, schema, and procedure name (if procedures are supported).
 - f. All `StoredProcedure` implementations' constructors. For example, `QSEmptyProcedure()` calls the super constructor for `StoredProcedure` with a null schema.

Appendix E: Java 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.

For the JavaQuickstart sample driver, a configuration file is used to enable this server-specific behavior. When the driver's server flag is set, the key/value pairs in the configuration file augment the connection settings that are passed in during a connection.

To set the JavaQuickstart sample driver up as a server:

1. Build the *QuickstartJNIDSI* using a server configuration (i.e. *Debug_Server* or *Release_Server*). This should build the server executable.
2. Build the *JavaQuickstart* DSII using the ANT build script, as you would for a standalone driver.
3. Edit the file *qsconfig.properties* so that the server flag is set to true and the [INSTALL_DIRECTORY] is replaced to point to the correct DBF location. It should include the following string values:
 - a. *SERVER=true*
 - b. *DBF=[INSTALL_DIRECTORY]||Examples||Databases||Text* (Note that the double slashes are needed in the path since the values are read in as Java Strings and the backslashes need to be escaped.)
4. Place the *qsconfig.properties* in the same directory as the *JavaQuickstart.jar* file.
5. Ensure that the JNIConfig is configured correctly, as you would for a standalone driver.

On Linux, to set the JavaQuickstart sample driver up as a server you need to:

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


```
BUILDSERVER=exe make -f QuickstartJNIDSI.mak debug
```
2. Build the *JavaQuickstart* DSII using the ANT build script, as you would for a standalone driver.
3. Edit the file *qsconfig.properties* so that the server flag is set to true and replace the DBF with the correct path. It should include the following string values:


```
SERVER=true
DBF=[INSTALL_DIRECTORY]/Examples/Databases/Text
```

4. Place the *qsconfig.properties* in the same directory as the *JavaQuickstart.jar* file.
5. Ensure that the JNIConfig is configured correctly, as you would for a standalone driver.

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.