



SimbaEngine X SDK X version
10.2

Build a Java ODBC Driver in 5
Days

Last Revised: October 2022

Simba Technologies Inc.



Copyright ©2019 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.

Trademarks

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

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

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

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

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

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

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

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

All other trademarks are trademarks of their respective owners.

Contact Us

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

www.simba.com

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

Fax +1 (604) 633-0004

Information and product sales: solutions@simba.com

Technical support: support@simba.com

Follow us on Twitter:

[@simbatech](https://twitter.com/simbatech)

Table of Contents

Introduction	1
About SimbaEngine	1
About the JavaQuickstart sample driver.....	2
Overview	3
Day One – Windows Instructions.....	4
Install SimbaEngine	4
Components of the JavaQuickstart example driver	4
Set environment variables	4
Build the native C++ component (QuickstartJNIDSI).....	4
Build the Java component (JavaQuickstart)	5
Update the registry	6
Enable Debugging	6
Examine the registry keys for SimbaEngine.....	7
Test the data source	8
Set up a new project to build your own ODBC driver	9
Build the native C++ component.....	9
Build the Java component.....	10
Update the registry	11
Test your new data source.....	12
Day One – Linux Instructions	12
Install SimbaEngine	12
Components of the JavaQuickstart example driver	13
Build the native C++ component (QuickstartJNIDSI).....	13
Build the Java component (JavaQuickstart)	14
Configure the ODBC data source and ODBC driver	15
Test the data source	18
Build your new ODBC driver	18
Configure an ODBC data source and ODBC driver	19
Test your new data source.....	20
Day Two.....	20
Find or create the Java Virtual Machine	20
Set the driver properties	21
Set the logging details	21
Check the connection settings	21
Establish a connection.....	22
Update the configuration file	22
Day Three	22
Create and return metadata sources	23
Day Four	24
Open a table.....	24
Return the catalog, schema and table names for your table	25
Return the columns defined for your table	25
Data Retrieval.....	26
Close the connection	26
Day Five.....	26
Assign a component ID.....	26

Update the error messages	27
Extend the ExceptionBuilder class.....	27
Configure the error messages	27
Create a driver configuration dialog	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
32-Bit Drivers on 32-Bit Windows	30
32-Bit Drivers on 64-Bit Windows	31
64-Bit Drivers on 64-Bit Windows	32
Appendix C: Data Retrieval.....	35
Appendix D: How to Add Schema Support	37
Handling DSI_SCHEMAONLY_METADATA	37
Appendix E: Java Server Configuration	38
Third Party Licenses.....	40

Introduction

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

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

About SimbaEngine X SDK

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

You use SimbaEngine to create an executable file that will be accessed by common reporting applications and to access your data store when SimbaEngine executes an SQL statement. This executable file can be a Windows DLL, a Linux or Unix shared object, a stand-alone server, or some other form of executable. You create a custom-designed DSI implementation (DSII) that connects directly to your data source. Then, you create the executable by linking libraries from SimbaEngine with the DSI implementation that you have written. In the process, the project files or make files will link in the appropriate SimbaODBC and SimbaEngine libraries to complete the driver. In the final executable, the components from SimbaEngine take responsibility for meeting the data access standards while your custom DSI implementation takes responsibility for accessing your data store and translating it to the DSI API.

About the JavaQuickstart sample driver

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

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

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

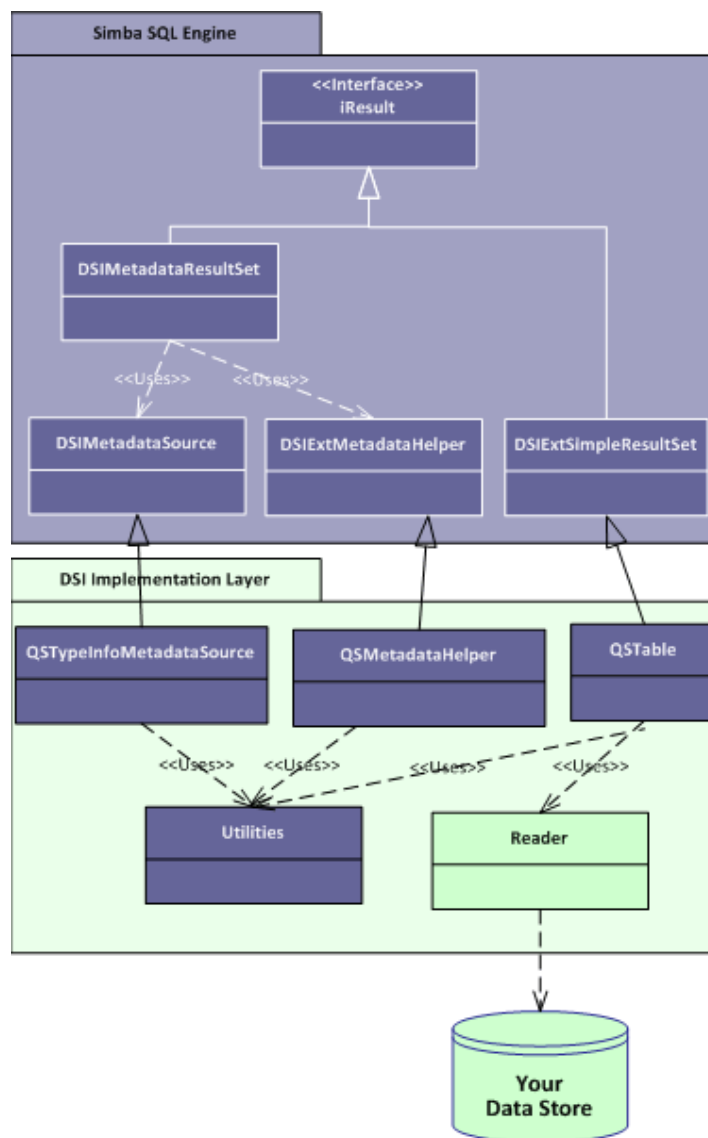


Figure 1: Design pattern for a DSI implementation.

There is a circular pattern of class relationships, headed by `IResult` and anchored by `QSUtilities`. The `IResult` class is responsible for retrieving column data and maintaining a cursor across result rows and the `QSUtilities` class contains platform-specific utility functions.

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

As a starting point, to make your driver work properly with Microsoft Excel, you can add metadata access by implementing the `QSTypeInfoMetadataSource` class and using the `DSIExtMetadataHelper` class. The `DSIExtMetadataHelper` class is responsible for iterating through tables and stored procedures so the engine can generate catalog function metadata.

Overview

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

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

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

Day One – Windows Instructions

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 ODBC driver.

Install SimbaEngine X SDK

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

1. Run the SimbaEngine setup executable and follow the installer's instructions.

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

Components of the JavaQuickstart example driver

Java ODBC drivers that are built using SimbaEngine have two components:

- The native C++ component (QuickstartJNIDSI) finds or creates an instance of the Java Virtual Machine (JVM) and provides the Java driver implementation name to the bridge between C++ and Java (SimbaJNIDSI).
- The Java component (JavaQuickstart) is the DSII.

Set environment variables

1. Ensure that the environment variable, JAVA_HOME, is pointing to the root of your JDK. To do this, go to your computer's System Properties configuration window, click the Advanced tab and then click Environment Variables. If JAVA_HOME is not listed in the System variables list, click New and in the Variable name field, type JAVA_HOME and in the Variable value field, type the path of the java home directory. For example, it might be something like this: C:\Program Files\Java\jdk1.7.0_09.
2. Ensure that the PATH environment variable includes the path to the jvm.dll. If the PATH variable needs to be updated, click **Edit** and then add the path to the jvm.dll file to the end of the path. It will look something like this: C:\Program Files\Java\jdk1.7.0_09\jre\bin\server\.

Note: There is a difference between the 32-bit and 64-bit jvm.dll files and you must set the path to the correct one.

Build the native C++ component (QuickstartJNIDSI)

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

1. Launch Microsoft Visual Studio.
2. Click **File > Open > Project/Solution**.
3. Navigate to
`[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\JavaQuickstart\Source\JavaQuickstartJNIDSI` and then open the `QuickstartJNIDSI_VS2012.sln` file. This solution file contains the `QuickstartJNIDSI`, which is the driver's native component. The default `[INSTALL_DIRECTORY]` is `C:\Simba Technologies`.
4. Click **Build > Configuration Manager** and make sure that the active solution configuration is "Debug_MTDLL" and then click **Close**.
5. Click **Build > Build Solution** or press F7 to build the driver. This will build the `Debug_MTDLL` version of the driver and place it in the following location for 32-bit drivers:

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

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

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

Build the Java component (JavaQuickstart)

You can use ANT or tools that support ANT, such as the Eclipse IDE, to build the Java component. As an example, these instructions describe how to build `JavaQuickstart` using the Eclipse IDE.

1. Launch the Eclipse IDE.
 Note: You can download the Eclipse IDE for Java Developers from www.eclipse.org/downloads.
2. Import the `JavaQuickstart` project as an existing project into your workspace. To do this, click `File > Import`. In the Import window, click `General > "Existing Projects into Workspace"` and then click `Next`. Choose the "Select root directory" option and then click `Browse` to navigate to
`[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\JavaQuickstart\Source\JavaQuickstartDSII`. Then click `Finish`.
3. Click `Project > Properties > Java Build Path > Libraries`. Select the `SIMBAENGINE_DIR` entry. Click `Edit` and then click `Variable`. The Variable Selection window opens. Click `New`. In the New Variable Entry window that opens, in the Name field, type `SIMBAENGINE_DIR` and in the Path field, type the path of the `DataAccessComponents` folder of your installed SimbaEngine. For example, it might be something like this: `C:\Simba Technologies\SimbaEngineSDK\10.1\DataAccessComponents`. Click `OK`.
4. A message is displayed, saying that the classpath variables have changed and that a full rebuild is recommended. Click `No`.
5. Click `OK` to close the properties windows.
6. Click `Run > External Tools > External Tools Configurations`.

7. In the External Tools Configurations window, double-click Ant Build. A setup page for the new Ant build configuration is displayed.
8. In the Name field, type JavaQuickstart.
9. On the "Main" tab, in the Buildfile section, click Browse Workspace.
10. In the Choose Location window, click JavaQuickstartBuilder.xml and then click OK.
11. Click Apply.
12. Click Run. This will build the JavaQuickstart driver using Apache Ant and it will place it in the location: [INSTALL_DIRECTORY]\Examples\Source\JavaQuickstart\Lib.

Update the registry

To update the registry keys, do the following:

1. In Microsoft Visual Studio, click **File > Open > File** and navigate to [INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\JavaQuickstart\Source.
2. For 32-bit Windows, open SetupMyJavaQuickstartDSII-32on32.reg.

For a 32-bit ODBC driver on 64-bit Windows, open SetupMyJavaQuickstartDSII-32on64.reg.

For a 64-bit ODBC driver on 64-bit Windows, open SetupMyJavaQuickstartDSII-64on64.reg.

3. In the file, replace [INSTALL_DIRECTORY] with the path to the installation directory. In the path, you must enter double backslashes. For example, by default, the samples are installed to "C:\Simba Technologies" so in that case, you would replace all instances of [INSTALL_DIRECTORY] with C:\\Simba Technologies.
4. 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.
5. Beside the line that starts with "Driver" = verify that the path to the driver dll file is correct.
6. Click **Save** and then close the file.
7. Double-click the registry file that you just modified. A message is displayed that says that the keys and values have been successfully added to the registry.

Enable Debugging

To enable debugging for the JavaQuickstart sample driver:

1. In the Registry file that you edited in the section *Update the registry* on page 6, edit the value data for the JNIConfig value so that the value for -Djava.class.path is correct.

2. Save, close, and then double-click the Registry file that you modified in step 1. A message appears indicating that the keys and values have been successfully added to the Registry.

Examine the registry keys for SimbaEngine X SDK

SimbaEngine uses the following registry keys that define Data Source Names (DSNs) and driver locations:

- **ODBC Data Sources** - lists each DSN/driver pair
- **JavaQuickstartDSII** - defines the Data Source Name (DSN). Used by the ODBC Driver Manager to connect your driver to your database.
- **ODBC Drivers** - lists the drivers that are installed
- **JavaQuickstartDSIIDriver** - defines the driver and its setup location. The ODBC Driver Manager uses this key.

To view the registry keys, do the following:

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

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

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

For 32-bit drivers on 64-bit Windows:

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

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

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

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

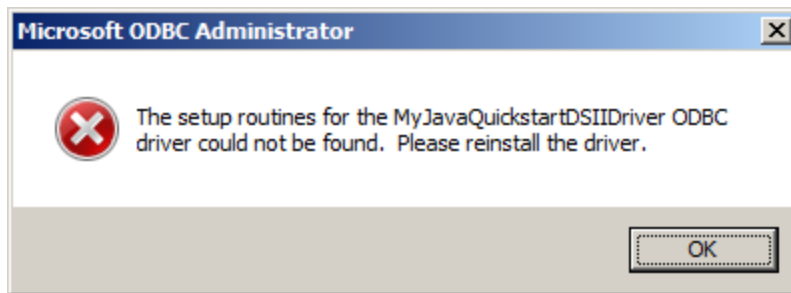
For 32-bit drivers on 64-bit Windows:

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

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

Note: Registry keys for 32-bit and 64-bit ODBC drivers are installed in different areas of the Windows registry. See Appendix B: Windows Registry 32-Bit vs. 64-Bit on page 30 for more information.

Note: You cannot use the ODBC Data Source Administrator to configure this data source because a configuration dialog has not been provided for the JavaQuickstart driver. If you try to use the ODBC Data Source Administrator, you will see the following error message:



Test the data source

To test the data source that we have created, you can use any ODBC application, such as, for example, Microsoft Excel, Microsoft Access or ODBCTest. In this section, we will use the ODBC Test tool, which is available in the Microsoft Data Access (MDAC) 2.8 Software Development Kit (SDK). To download SimbaEngine, visit the following Microsoft Web site:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=5067faf8-0db4-429a-b502-de4329c8c850&displaylang=en>

Note: Before you test the data source, you must ensure that the `jvm.dll` file is in your path. This is described in the section, "Set environment variables".

1. Start the ODBC Test tool. By default, the ODBC Test application is installed in the following folder: `C:\Program Files (x86)\Microsoft Data Access SDK 2.8\Tools\`

Navigate to the folder that corresponds to your machine's architecture (amd64, ia64 or x86) and then click `odbcte32.exe` to launch the ANSI version or click `odbct32w.exe` to launch the Unicode version.

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

2. In the ODBC Test tool, select **Conn > Full Connect**.
The Full Connect window opens.
3. Select your Data Source from the list of data sources and then click **OK**.
If you do not see your data source in the list, make sure that you are running the version of the ODBC Test tool that corresponds to the version of the data source that you created. In other words, if you created a 32-bit data source then you should be using the 32-bit version of the ODBC Test tool.
4. When the tool connects to the data source, you will see the message, "Successfully connected to DSN 'MyJavaQuickstartDSII'".

Set up a new project to build your own ODBC driver

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

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

1. In your Windows Explorer window, copy the `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\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. For the rest of this tutorial, when you see `<YourProjectName>` in the instructions, replace this with the name you choose for this directory which is also the name of your project.
2. Open your new directory then open the `Source` directory.
3. Rename the `JavaQuickstartJNIDSI` directory to `<YourProjectName>JNIDSI`.
4. Rename the `JavaQuickstartDSII` directory to `<YourProjectName>DSII`.
5. Go to the `<YourProjectName>JNIDSI` directory and then right-click the `QuickstartJNIDSI_VS2012.sln` file.
6. Select **Open with > Microsoft Visual Studio Version Selector**.
7. In the Microsoft Visual Studio menu, click **View > Solution Explorer**.
8. Using the Solution Explorer, rename the **QuickstartJNIDSI_VS2012** solution to **<YourProjectName>JNIDSI_VS2012**.
9. Rename the **QuickstartJNIDSI** project to **<YourProjectName>JNIDSI**.
10. Change the branding in `Simba::JNIDSI::SetConfigurationBranding()` in `main_windows.cpp`.
11. Click **File > Save All**.
12. In Windows Explorer, go to the `<YourProjectName>DSII` directory and then rename the `JavaQuickstartBuilder.xml` file to `<YourProjectName>Builder.xml`. This is the Apache Ant builder file (`.xml`) for your new ODBC driver. Using a text editor, open the file and replace every instance of "JavaQuickstart" and "JavaQS" in the source code with the name of your new ODBC driver. Next, update the copyright information for the "doc" target. Then save and close the file.
13. Open the `.project` file in a text editor and replace the `JavaQuickstart` within the `<name>` tags with `<YourProjectName>`. Save and close the file.

Build the native C++ component

1. In Microsoft Visual Studio, click **Build > Build Solution** or press F7 to build the driver.

When you build your new project, "TODO" messages appear in the Output window along with the build information. If the Output window is not displayed automatically, you can open it by selecting **Debug > Windows > Output**.

```

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

```

Build the Java component

You can use ANT or tools that support ANT, such as the Eclipse IDE, to build the Java component. As an example, these instructions describe how to build JavaQuickstart using the Eclipse IDE.

1. With the Eclipse IDE, click **File > Import**. In the Import window, click **General > "Existing Projects into Workspace"** and then click **Next**. Choose the "Select root directory" option and then click **Browse** to navigate to `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\<YourProjectName>\Source\<YourProjectName>DSII`. Then click **Finish**.
2. To see your new Java project in the Project Explorer window, click **Window > Show View > Project Explorer**.
3. Click **Run > External Tools > External Tools Configurations**.
4. In the External Tools Configurations window, double-click **Ant Build**. A setup page for the new Ant build configuration is displayed.
5. In the **Name** field, type `<YourProjectName>`.
6. On the "Main" tab, in the Buildfile section, click **Browse Workspace**.
7. In the Choose Location window, click `<YourProjectName>Builder.xml` and then click **OK**.
8. Click **Apply**.
9. Still in the Ant build configuration, switch to the **Environment** tab. Add an environment variable called `SIMBAENGINE_DIR` with the value `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\DataAccessComponents`. Click **Apply**.
10. Click **Run**. This will build the driver using Apache Ant.

Search your Java workspace for 'TODO' to find the following comments that mark locations where changes to your driver code 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 visit each “TODO” and modify the source code.

Update the registry

You must update the Windows registry to add the information for your new ODBC driver before you will be able to test it.

To update the registry keys, do the following:

1. In Microsoft Visual Studio, click **File > Open > File** and navigate to `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\\Source`.
2. For 32-bit Windows, open `SetupMyJavaQuickstartDSII-32on32.reg`.

For a 32-bit ODBC driver on 64-bit Windows, open `SetupMyJavaQuickstartDSII-32on64.reg`.

For a 64-bit ODBC driver on 64-bit Windows, open `SetupMyJavaQuickstartDSII-64on64.reg`.

3. In the file, replace `[INSTALL_DIRECTORY]` with the path to the installation directory. In the path, you must enter double backslashes. For example, by default, the samples are installed to “`C:\Simba Technologies`” so in that case, you would replace all instances of `[INSTALL_DIRECTORY]` with `C:\\Simba Technologies`.
4. To enable debugging, update the value data for the `JNIConfig` value so that the value for `-Djava.class.path` is correct.
5. Next, update the ODBC Data Sources section to add your new data source. Under the `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources]` section, change `"MyJavaQuickstartDSII"="MyJavaQuickstartDSIIDriver"` to the name of your new data source and new driver. For example, `"<YourProjectName>DSII"="<YourProjectName>DSIIDriver"`
6. Then, modify the data source definition for that data source. Change the line that says `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\MyJavaQuickstartDSII]` so that it contains your new data source name. For example, `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\<YourProjectName>DSII]`
7. Beside the line that starts with `"Driver"`, update the path to the dll file.
8. Update the ODBC Drivers section to add your new driver. Under the `[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers]` section,

change "MyJavaQuickstartDSIIDriver"="Installed" to match the name of your new driver. For example, "<YourProjectName>DSIIDriver"="Installed"

9. Modify the driver definition for that driver. Change the line that says
`[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\MyJavaQuickstartDSIIDriver]`
 so that it contains your new driver name. For example,
`[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\<YourProjectName>DSIIDriver]`
10. Beside the line that starts with "Driver", update the path to the dll file.
11. Click **Edit > Find and Replace > Quick Replace**. Then, replace "JavaQuickstart" in the whole file with the name of your new driver.
12. Click **Save** and then close the file.
13. In the Registry Editor (`regedit.exe`), click **File > Import**, navigate to the registry file that you just modified and then click **Open**.
 A message is displayed that says that the keys and values have been successfully added to the registry.

Note: You cannot use the ODBC Data Source Administrator to configure this data source.

Test your new data source

1. Start the ODBC Test tool. By default, the ODBC Test application is installed in the following folder: `C:\Program Files (x86)\Microsoft Data Access SDK 2.8\Tools\`
 Navigate to the folder that corresponds to your driver's architecture (amd64, ia64 or x86) and then click `odbcte32.exe` to launch the ANSI version or click `odbct32w.exe` to launch the Unicode version. It is important to run the correct version of the ODBC Test tool for ANSI or Unicode and 32-bit or 64-bit.
2. In the ODBC Test tool, select **Conn > Full Connect**.
 The Full Connect window opens.
3. Select your Data Source from the list of data sources and then click **OK**.
 If you do not see your data source in the list, make sure that you are running the version of the ODBC Test tool that corresponds to the version of the data source that you created. In other words, if you created a 32-bit data source then you should be using the 32-bit version of the ODBC Test tool.
4. When the tool connects to the data source, you will see the message "Successfully connected to DSN '<YourProjectName>DSII'".

Day One – Linux Instructions

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

Install SimbaEngine X SDK

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

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. The "*" in the file name represents a string of characters that represent the build number and platform. For example, the file name might look something like this:
SimbaEngineSDK_Eval_Linux-x86_10.1.0.1000.tar.gz

1. Open a command prompt.
2. Change to the directory where you want to install SimbaEngine. Later in the instructions, we will refer to this as <installdir>.
3. Copy the SimbaEngineSDK*.tar.gz file to that directory.
4. To uncompress the file, type: `gunzip SimbaEngineSDK*.tar.gz`
5. To extract the tar file, type: `tar -xvf SimbaEngineSDK*.tar`

Components of the JavaQuickstart example driver

Java ODBC drivers that are built using SimbaEngine have two components:

- The native C++ component (QuickstartJNIDSI) finds or creates an instance of the Java Virtual Machine (JVM) and provides the Java driver implementation name to the bridge between C++ and Java (SimbaJNIDSI).
- The Java component (JavaQuickstart) is the DSII.

Build the native C++ component (QuickstartJNIDSI)

On Linux and UNIX platforms, the sample drivers use makefiles instead of Visual Studio solution files.

To build the SimbaEngine Quickstart sample driver, the steps are as follows:

1. Ensure that the environment variable, `JAVA_HOME`, is pointing to the root of your JDK. To do this open a Terminal window and type the following command:

```
echo $JAVA_HOME
```

2. If no output is displayed, set the `$JAVA_HOME` environment variable. For example, you might add something like this to your `.bashrc` file:

```
export JAVA_HOME=/usr/java/jdk1.6.0_31/
```

3. Ensure that the `PATH` environment variable includes the path to java. If the `PATH` variable needs to be updated, you can modify your `.bashrc` file.

4. Set the `SIMBAENGINE_DIR` environment variable. For example, you might add the following line to your `.bashrc` file:

```
export  
SIMBAENGINE_DIR=<installdir>/SimbaEngineSDK/10.1/DataAccessComponents
```

5. Set the `SIMBAENGINE_THIRDPARTY_DIR` environment variable by typing:

```
export
```

```
SIMBAENGINE_THIRDPARTY_DIR=<installdir>/SimbaEngineSDK/10.1/DataAccessComponents/ThirdParty
```

6. In the command above, replace <installdir> with the directory where you installed the SimbaEngine files.

7. Type the following command to change to the directory that contains the makefile:

```
cd <installdir>/SimbaEngineSDK/10.1/Examples/Source/JavaQuickstart/Source
```

8. Type the following command to run the makefile for the debug target:

```
./mk.sh MODE=debug
```

For more build options, see the SimbaEngine X SDK Developer Guide. If you are using XCode, by default, the makefile detects the latest version of XCode on your machine. Optionally, you can specify a different version of XCode using the environment variable DEVELOPER_DIR.

Build the Java component (JavaQuickstart)

1. Launch the Eclipse IDE.
Note: You can download the Eclipse IDE for Java Developers from www.eclipse.org/downloads.
2. Import the JavaQuickstart project as an existing project into your workspace. To do this, click **File > Import**. In the Import window, click **General > "Existing Projects into Workspace"** and then click **Next**. Choose the "Select root directory" option and then click **Browse** to navigate to [INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\JavaQuickstart\Source\JavaQuickstartDSII. Then click **Finish**.
3. Select the JavaQuickstart project in the Package Explorer.
4. Click **Project > Properties > Java Build Path > Libraries**. Select the SIMBAENGINE_DIR entry. Click **Edit** and then click **Variable**. The Variable Selection window opens. Click **New**. In the New Variable Entry window that opens, in the Name field, type SIMBAENGINE_DIR and in the Path field, type the path of the DataAccessComponents folder of your installed SimbaEngine. For example, it might be something like this:
[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\DataAccessComponents. Click **OK**.
5. A message is displayed, saying that the classpath variables have changed and that a full rebuild is recommended. Click **No**.
6. Click **OK** to close the properties windows.
7. Click **Run > External Tools > External Tools Configurations**.
8. In the External Tools Configurations window, double-click Ant Build. A setup page for the new Ant build configuration is displayed.
9. In the Name field, type JavaQuickstart.
10. On the "Main" tab, in the Buildfile section, click Browse Workspace.
11. In the Choose Location window, click JavaQuickstartBuilder.xml and then click **OK**.
12. Click **Apply**.

13. Still in the Ant build configuration, switch to the **Environment** tab. Add an environment variable called `SIMBAENGINE_DIR` with the value `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\DataAccessComponents`.
14. Click **Apply**.
15. Click **Run**. This will build the JavaQuickstart driver using Apache Ant and it will place it in the location: `[INSTALL_DIRECTORY]\Examples\Source\JavaQuickstart\Lib`.

Configure the ODBC data source and ODBC driver

ODBC driver managers use configuration files to define and configure ODBC data sources and drivers. The `odbc.ini` file is used to define ODBC data sources and the `odbcinst.ini` file is used to define ODBC drivers.

Location of the ODBC configuration files

The value of the `$ODBCINI` and `$ODBCSYSINI` environment variables specify the location of the configuration files. If these environment variables are not set, it is assumed that the configuration files will be in the user's home directory and the default filename must be used (`.odbc.ini` and `.odbcinst.ini`).

Optionally, if you decide to put the configuration files somewhere other than the user's home directory, set the environment variables by typing a command similar to the following example:

```
export ODBCINI=/usr/local/odbc/myodbc.ini
export ODBCSYSINI=/usr/local/odbc/myodbcinst.ini
```

Samples of the configuration files are provided in the following directory:

```
<installdir>/SimbaEngineSDK/10.1/Documentation/Setup
```

Configure an ODBC data source

ODBC Data Sources are defined in the `.odbc.ini` configuration file.

To configure a data source:

1. To see if the `.odbc.ini` file already exists in your home directory, type the following command:

```
ls -al ~ | grep .odbc.ini
```

If the file exists, you will see something like this:

```
-rw-rw-r-- 1 employee employee 1379 Oct 23 14:56 .odbc.ini
```

If the file doesn't exist, then the command will not return anything. In this case, copy the `odbc.ini` file from the samples directory by typing:

```
cp <installdir>/SimbaEngineSDK/10.1/Documentation/Setup/odbc.ini
~/.odbc.ini
```

2. Open the `~/.odbc.ini` configuration file in a text editor. To open the file, you may need to configure your text editor to show hidden files.

3. Make sure there is an entry in the [ODBC Data Sources] section that defines the data source name (DSN). The [ODBC Data Sources] section is used to specify the available data sources.

```
[ODBC Data Sources]
JavaQuickstartDSII=JavaQuickstartDSIIDriver
```

4. Make sure there is a section with a name that matches the data source name (DSN). This section will contain the configuration options. They are specified as key-value pairs.

```
[JavaQuickstartDSII]
Description=My SimbaEngine JavaQuickstart DSII
DBF=<installdir>/SimbaEngineSDK/10.1/Examples/Databases/Quickstart/
Driver=<installdir>/SimbaEngineSDK/10.1/Examples/Source/JavaQuickstart/
Bin/Linux_x8664/libQuickstartJNIDSI_debug.so
```

Define an ODBC driver

ODBC Drivers are defined in the `.odbcinst.ini` configuration file. This configuration is optional because drivers can be specified directly in the `.odbc.ini` configuration file as discussed in the previous section.

To define a driver:

1. To see if the `.odbcinst.ini` file exists in your home directory, type the following command:

```
ls -al ~ | grep .odbcinst.ini
```

If the file exists, you will see something like this:

```
-rw-rw-r-- 1 employee employee 2272 Oct 23 15:30 .odbcinst.ini
```

If the file doesn't exist, then the command will not return anything. In this case, copy the `odbc.ini` file from the samples directory by typing:

```
cp <installdir>/SimbaEngineSDK/10.1/Documentation/Setup/odbcinst.ini
~/odbcinst.ini
```

2. Open the `~/odbcinst.ini` configuration file in a text editor.
3. Add a new entry to the [ODBC Drivers] section. The [ODBC Drivers] section is used to specify the available drivers. Type driver name and the value "Installed". This driver name should be used for the "Driver" value in the data source definition instead of the driver shared library name. For example, it might look something like this:

```
[ODBC Drivers]
JavaQuickstartDSIIDriver=Installed
```

4. Add a new section with a name that matches the new driver name. This section will contain the configuration options. They are specified as key-value pairs.

```
[JavaQuickstartDSIIDriver]
Driver=<installdir>/SimbaEngineSDK/10.1/Examples/Source/JavaQuickstart/
Bin/Linux_x8664/libQuickstartJNIDSI_debug.so
```

Configure the Simba JavaQuickstart ODBC Driver

1. To see if the `.simba.javaquickstart.ini` file already exists in your home directory, type the following command:

```
ls -al ~ | grep .simba.javaquickstart.ini
```

2. If the file doesn't exist, then the command will not return anything. In this case, copy the file from the samples directory by typing:


```
cp <installdir>/SimbaEngineSDK/10.1/Documentation/Setup/.simba.javaquickstart.ini ~/.simba.javaquickstart.ini
```
3. Open the `~/.simba.javaquickstart.ini` configuration file in a text editor.
4. Edit the `DriverManagerEncoding` setting.

If you are using the "iODBC" ODBC Driver Manager set the `DriverManagerEncoding` setting to **UTF-32**.

-or-

If you are using the "unixODBC" ODBC Driver Manager, you will need to check which setting to use. Type `odbc_config --cflags` at a command prompt. If you see the "DSQL_WCHART_CONVERT" flag, then set the `DriverManagerEncoding` setting to **UTF-32**. Otherwise, set it to **UTF-16**.

For more information about your ODBC driver manager, consult your system administrator or your ODBC Driver Manager documentation.
5. Edit the `ErrorMessagesPath` setting to replace `[INSTALLDIR]` with your install directory.
6. 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 look something like this:

```
ODBCInstLib=<driver manager dir>/lib/libiodbcinst.so (notice the 'i' after the lib)
```

For unixODBC this would be:

```
ODBCInstLib=<driver manager dir>/lib/libodbcinst.so
```
7. Save the file.

For more information about how to configure data sources under Linux, Unix and MacOSX, please refer to the SimbaEngine X SDK Developer Guide.

Enable Debugging

To enable debugging for the JavaQuickstart sample driver:

1. In the `.simba.javaquickstart.ini` file that you edited in the section *Configure the Simba JavaQuickstart ODBC Driver* on page 16, update the `JNIConfig` line to the say the following:

```
JNIConfig=-
Djava.class.path=<installdir>/SimbaEngineSDK/10.1/Examples/Source/
JavaQuickstart/Lib/JavaQuickstart.jar
```

OR

To enable remote debugging of your DSII, update the `JNIConfig` line to something like this:

```
JNIConfig=-
Djava.class.path=<installdir>/SimbaEngineSDK/10.1/Examples/Source/
JavaQuickstart/Lib/JavaQuickstart.jar|-Xdebug|-
Xrunjdpw:transport=dt_socket,address=localhost:8000,suspend=n,server=y
```

2. Save the file that you edited in step 1.

Test the data source

Prerequisites:

- You must have the International Components for Unicode (ICU) libraries in the `LD_LIBRARY_PATH` environment variable.

To add the 32-bit ICU libraries, type the following at the command line:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:[INSTALLDIR]/SimbaEngineSDK/10.1/
DataAccessComponents/ThirdParty/icu/53.1/centos5/gcc4_4/release32/lib:
[INSTALLDIR]/SimbaEngineSDK/10.1/DataAccessComponents/ThirdParty/openssl/
1.0.1/centos5/gcc4_4/release32/lib
```

To add the 64-bit ICU libraries, type the following at the command line:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:[INSTALLDIR]/SimbaEngineSDK/10.1/
DataAccessComponents/ThirdParty/icu/53.1/centos5/gcc4_4/release64/lib:
[INSTALLDIR]/SimbaEngineSDK/10.1/DataAccessComponents/ThirdParty/openssl/
1.0.1/centos5/gcc4_4/release64/lib
```

- You must have the path to the `libjvm.so` file in the `LD_LIBRARY_PATH` environment variable. For example, the file might be in a location like this:
`/usr/java/jdk1.6.0_31/jre/lib/amd64/server.`
- You must have a Driver Manager such as `iODBC` or `unixODBC` installed. For more detailed information on Driver Managers and testing, please refer to the *SimbaEngine X SDK Developer Guide*.

One way to test your data source is to use the test utility, `iodbctest`, which is included with the `iODBC` Driver Manager:

- At the command prompt, type the following command: `iodbctest`
- At the prompt that says “Enter ODBC connect string”, type `?` to show the list of DSNs and Drivers.
- In the list, you should see your `JavaQuickstartDSII` DSN.
- To connect to your data source, type the following command:

```
DSN=JavaQuickstartDSII
```

A prompt that says “SQL>” appears.

- Type a SQL command to query your database. For example, `SELECT * FROM PRODUCT.`

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

Build your new ODBC driver

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

1. Copy the JavaQuickstart directory to a new directory that will be the top-level directory for your new project and DSI implementation files. For example, you could do it like this:

```
cp -R <installdir>/SimbaEngineSDK/10.1/Examples/Source/JavaQuickstart
<installdir>/SimbaEngineSDK/10.1/Examples/Source/MyJavaQuickstart
```

Note: It is very important that you take this step to create your own directory because there may be times, for debugging purposes, that you will need to see if the same error occurs using the sample drivers. If you have modified the sample drivers, this will not be possible.

2. Open your new directory then open the `Makefiles` directory and rename the `JavaQuickstart.mak` file in it. For example, you could type `mv JavaQuickstart.mak MyJavaQuickstart.mak`.
3. Then, rename the `.depend` file that is located in the `Makedepend` directory.
4. Open your new directory then open the `Source` directory. Open the `Makefile` file and replace “JavaQuickstart” project name in the source code with the name of your new ODBC driver. Then save and close the file.
5. Change to the following directory:

```
<installdir>/SimbaEngineSDK/10.1/Examples/Source/MyJavaQuickstart/Source
```
6. To run the makefile for the debug target, type the following command:

```
./mk.sh MODE=debug
```

Configure an ODBC data source and ODBC driver

1. Open the `.odbc.ini` configuration file in a text editor.
2. Make sure there is an entry in the [ODBC Data Sources] section that defines the data source name (DSN).

```
[ODBC Data Sources]
MyJavaQuickstartDSII=MyJavaQuickstartDSIIDriver
```

3. Make sure there is a section with a name that matches the data source name (DSN).

```
[MyJavaQuickstartDSII]
Description=My SimbaEngine DSII
DBF=<installdir>/SimbaEngineSDK/10.1/Examples/Databases/Quickstart/
Driver=<installdir>/SimbaEngineSDK/10.1/Examples/Source/MyJavaQuickstart/
Bin/Linux_x8664/libQuickstartJNIDSI_debug.so
```

4. Open the `.odbcinst.ini` configuration file in a text editor.
5. Add a new entry to the [ODBC Drivers] section. For example, it might look something like this:

```
[ODBC Drivers]
MyJavaQuickstartDSIIDriver=Installed
```

6. Add a new section with a name that matches the new driver name.

```
[MyJavaQuickstartDSIIDriver]
```



```
Driver=<installdir>/SimbaEngineSDK/10.1/Examples/Source/MyJavaQuickstart/
Bin/Linux_x8664/libQuickstartJNIDSI_debug.so
```

7. Copy the `.simba.javaquickstart.ini` file in your home directory so it has the name `.simba.myquickstart.ini`, and then update settings in `.simba.javaquickstart.ini` for your own ODBC driver as described in *Configure the Simba JavaQuickstart ODBC Driver* on page 16 as needed.
8. To enable debugging, in the `.simba.javaquickstart.ini` file, update the `JNIConfig` line to use the path to your new jar file.

Test your new data source

One way to test your data source is to use the test utility, `iodbctest`, which is included with the iODBC Driver Manager:

1. At the command prompt, type the following command: `iodbctest`
2. At the prompt that says, "Enter ODBC connect string", type `?` to show the list of DSNs and Drivers.
3. In the list, you should see your `MyJavaQuickstartDSII` DSN.
4. To connect to your data source, type the following command:

```
DSN=MyJavaQuickstartDSII
```

A prompt that says "SQL>" appears.

5. Type a SQL command to query your database. For example, `SELECT * FROM PRODUCT.`
6. To quit `iodbctest`, at the prompt, type `quit.`

At this point, you have built and tested the Quickstart driver to make sure that your installation worked properly and that your development system is properly set up. In addition, you have created, built and tested your own copy of the JavaQuickstart Driver example that you will modify to work with your own data store.

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.

Find or create the Java Virtual Machine

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

1. In Visual Studio, in the C++ project, you will see the TODO messages in the Output window. Double click the TODO message to jump to the relevant section of code. If you cannot see the TODO messages, rebuild the solution by selecting **Build > Rebuild Solution**. If it is not already displayed, you can open the Output window by selecting **Debug > Windows > Output**.

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()`. 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.

Set the driver properties

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

1. In Eclipse, in your project, go to the TODO #1 message in the `QSDriver.java` file. Look at `setProperty()` where you will set up the general properties for your driver.
2. Change the `DSI_DRIVER_DRIVER_NAME` to the name of your new driver.

Set the logging details

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

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

1. Go to the TODO #2 message.
2. Change the driver log's file name.
3. Go to the TODO #3 message.
4. Change the connection log's file name.
5. Click **Save All**.

Note: 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.

For more information about how to enable logging, refer to the SimbaEngine X SDK Developer Guide.

Check the connection settings

TODO #4: Check Connection Settings. (QSConnection.java)

When the Simba ODBC layer is given a connection string from an ODBC-enabled application, the Simba ODBC layer parses the connection string into key-value pairs. Then, the entries in the connection string and the DSN are sent to `updateConnectionSettings()` function for validation.

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

2. The `updateConnectionSettings()` function should validate that the key-value pairs within the `requestMap` are sufficient to create a connection. Use the `verifyRequiredSetting()` or `verifyOptionalSetting()` utility functions to do this.
3. If any of the values received are invalid, you should throw an exception.
4. If there are no further entries required, simply leave the `responseMap` empty.

Establish a connection

```
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()`.

Update the configuration file

```
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 the 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 about setting up a Java driver as a server, please refer to Appendix E: Java Server Configuration.

You have now authenticated the user against your data store.

Day Three

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

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

- SQLTables
- SQLColumns

Create and return metadata sources

TODO #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 the following:

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

Handle DSI_TYPE_INFO_METADATA

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

1. When called with `TYPE_INFO`, QSDataEngine's `makeNewMetadataTable()` will return an instance of `QSTypeInfoMetadataSource()`.
2. The SimbaEngine JavaQuickstart Driver example exposes support for the following types:

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

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

Handle 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`, QSDataEngine's `makeNewMetadataTable()` should return `NULL`. Returning `NULL` will signal SimbaEngine that it should use the metadata helper class returned via QSDataEngine'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 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.

You can now retrieve type metadata from within your data store.

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

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 retrieving data.

Open a table

TODO #8: Open A Table.

(`QSDatabaseEngine.java`)

`QSDatabaseEngine`'s `openTable()` is the entry point where the Simba SQL Engine requests that tables involved in the query be opened.

1. Modify `QSDatabaseEngine`'s `openTable()` method to check that the supplied catalog, schema and table names are valid and correspond to a table defined in your data store. If they are not, you should return null to indicate that the table does not exist. If the inputs are valid, a new instance of `QSTable` is returned.

`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.

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

Next, make changes to `QSTable` to make it work with your data store.

Return the catalog, schema and table names for your table

- o `QSTable`'s `QSTable()`: The constructor must be modified to take in the catalog, schema and table names and save them in member variables.
- o `QSTable`'s `getCatalogName()`: Returns `Quickstart.CATALOG` (because it has only a single catalog);
- o `QSTable`'s `getSchemaName()`: Returns `null` (because it does not support schemas);
- o `QSTable`'s `getTableName()`: Returns `m_tableName`;

Return the columns defined for your table

- o `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 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 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 the section, Data Retrieval, for a brief overview of data retrieval.

Close the connection

- 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.

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

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.

Day Five

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

Assign a component ID

```
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.

Update the error messages

```
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.

Extend the ExceptionBuilder class

```
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.

Configure the error messages

```
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.

Create a driver configuration dialog

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

The C++ SimbaEngine Quickstart Driver project contains an example ODBC configuration dialog that you can look at, as an example. You can find the source under the Setup folder within the SimbaEngine Quickstart Driver project.

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

IMPORTANT: If you are using 64-bit Windows with 32-bit applications, you must use the 32-bit ODBC Data Source Administrator. You cannot access the 32-bit ODBC Data Source Administrator from the start menu or control panel in 64-bit Windows. Only the 64-bit ODBC Data Source Administrator is accessible from the start menu or control panel. On 64-bit Windows, to launch the 32-bit ODBC Data Source Administrator you must run `C:\WINDOWS\SysWOW64\odbcad32.exe`. See Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 29 for details.

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

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

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

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

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

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

PROBLEM: You cannot access the 32-bit ODBC Data Source Administrator from the start menu or control panel in 64-bit Windows.

SOLUTION: To create new 32-bit data sources or modify existing ones on 64-bit Windows you must run `C:\WINDOWS\SysWOW64\odbcad32.exe` (you may find it useful to put a shortcut to this on your desktop or Start menu if you access it frequently).

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

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

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

32-Bit Drivers on 32-Bit Windows

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

Data Source Names

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

- **JavaQuickstartDSII** which must include the following string values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release_MTDLL\QuickstartJNIDSI_MTDLL.dll`
 - **DBF:** `[INSTALL_DIRECTORY]\Examples\Databases\Quickstart`
 - **Description:** `Sample 32-bit SimbaEngine JavaQuickstart DSII`
- **JavaUltraLightDSII** which must include the following string values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release_MTDLL\UltraLightJNIDSI_MTDLL.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 32-bit Windows uses registry keys created in `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI`. Each key includes three string values to define the location of the **Driver**, its **Setup** location and the **Description** to help you clearly identify each registry key. The keys that are relevant to the C# examples discussed in this document are:

- **JavaQuickstartDSIIDriver** which includes the following key names and values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release_MTDLL\QuickstartJNIDSI_MTDLL.dll`

- **Setup:** Not included in this sample.
- **Description:** Sample 32-bit SimbaEngine JavaQuickstart DSII
- **JavaUltraLightDSIIDriver** which includes the following key names and values:
 - **Driver:** [INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release_MTDLL\UltraLightJNIDSI_MTDLL.dll
 - **Setup:** Not included in this sample.
 - **Description:** Sample 32-bit SimbaEngine 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 must also be added to it:

- **ODBC Drivers** which includes the following string values:
 - **JavaQuickstartDSIIDriver:** Installed
 - **JavaUltraLightDSIIDriver:** Installed

32-Bit Drivers on 64-Bit Windows

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

Data Source Names

To connect your driver to your database, the 32-bit ODBC Driver Manager on 64-bit Windows uses Data Source Name registry keys in

HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432NODE\ODBC\ODBC.INI. Each key includes 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 keys that are relevant to the examples discussed in this document are:

- **JavaQuickstartDSII** which must include the following string values:
 - **Driver:** [INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release_MTDLL\QuickstartJNIDSI_MTDLL.dll
 - **DBF:** [INSTALL_DIRECTORY]\Examples\Databases\Quickstart
 - **Description:** Sample 32-bit SimbaEngine JavaQuickstart DSII
- **JavaUltraLightDSII** which must include the following string values:
 - **Driver:** [INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release_MTDLL\UltraLightJNIDSI_MTDLL.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 keys that are relevant to the examples discussed in this document are:

- **JavaQuickstartDSIIDriver** which includes the following key names and values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release_MTDLL\QuickstartJNIDSI_MTDLL.dll`
 - **Setup:** Not included in this sample.
 - **Description:** Sample 32-bit SimbaEngine JavaQuickstart DSII
- **JavaUltraLightDSIIDriver** which includes the following key names and values:
 - **Driver:** `[INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release_MTDLL\UltraLightJNIDSI_MTDLL.dll`
 - **Setup:** Not included in this sample.
 - **Description:** Sample 32-bit SimbaEngine 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 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

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

- **JavaQuickstartDSII** which must include the following string values:
 - **Driver:** [INSTALL_DIRECTORY]\Examples\Builds\Bin\x64\Release\QuickstartJNIDSI.dll
 - **DBF:** [INSTALL_DIRECTORY]\Examples\Databases\Quickstart
 - **Description:** Sample 64-bit SimbaEngine JavaQuickstart DSII
- **JavaUltraLightDSII** which must include the following string values:
 - **Driver:** [INSTALL_DIRECTORY]\Examples\Builds\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 keys that are relevant to the examples discussed in this document are:

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

- Each `IMetadataSource` implementation of `getMetadata()`
- `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:

1. `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<Data Type>()` and `get<Data Type>()` 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(false)` when creating the `ColumnMetadata` for the column in `QSTable`'s `initializeColumns()`.

2. `offset`

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

3. `maxSize`

The maximum size (in bytes) that can be copied into the `in_data` parameter. For character

or binary data, copying data that is greater than this size can result in a data truncation warning or a heap-violation.

Appendix D: How to Add Schema Support

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

Handling DSI_SCHEMAONLY_METADATA

1. By default, `QSCConnection`'s `setDefaultProperties()` 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 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:
 - a. `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 to open the correct procedure.
 - b. `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).
 - c. 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. However, when the driver is a server, the settings cannot be retrieved directly because the DSN refers to the client instead of a specific driver. In addition, there would also be security concerns if a given client has control over server-specific settings. Therefore, to establish a connection when a driver is a server, the connection settings need to be augmented.

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 will 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:

```
SERVER=true
DBF=[INSTALL_DIRECTORY]\\Examples\\Databases\\Quickstart
```

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 and UNIX platforms, 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/Quickstart
```

4. Place the `qsconfig.properties` in the same directory as the `JavaQuickstart.jar` file.

5. Ensure that the JNIconfig is configured correctly. For further details, please see the SimbaClientServer User Guide.

Once you have configured the client and server, you should be able to connect to your data source.

Third Party Licenses

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2014 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

OpenSSL License

Copyright (c) 1998-2011 The OpenSSL Project. All rights reserved.

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

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

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

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay License

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)

All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are aheared to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

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

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

"This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"

The word 'cryptographic' can be left out if the rouines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:

"This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

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

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Expat License

"Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ""Software""), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ""AS IS"", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NOINFINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

Stringencoders License

Copyright 2005, 2006, 2007

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

All rights reserved.

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

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

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

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

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

This is the standard "new" BSD license:

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

dtoa License

The author of this software is David M. Gay.

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

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

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