SimbaEngine X version 10.1

Build a Java ODBC Driver for SQL-Based Data Sources in 5 Days

Last Revised: November 2016

Simba Technologies Inc.

Trademarks

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

Table of Contents

Introduction

This guide will show you how to create your own, custom ODBC driver using SimbaEngine. It will walk you through the steps to modify and customize the included JavaUltraLight 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 3.8 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

About SimbaEngine

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

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

About the JavaUltraLight sample driver

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

The JavaUltraLight 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.

Implementation begins with the creation of a DSIDriver class which is responsible for constructing a DSIEnvironment. This in turn is used to construct a connection object (DSIConnection implementation) which can then be used for constructing statements (DSIStatement implementations). This is summarized in Figure 1:

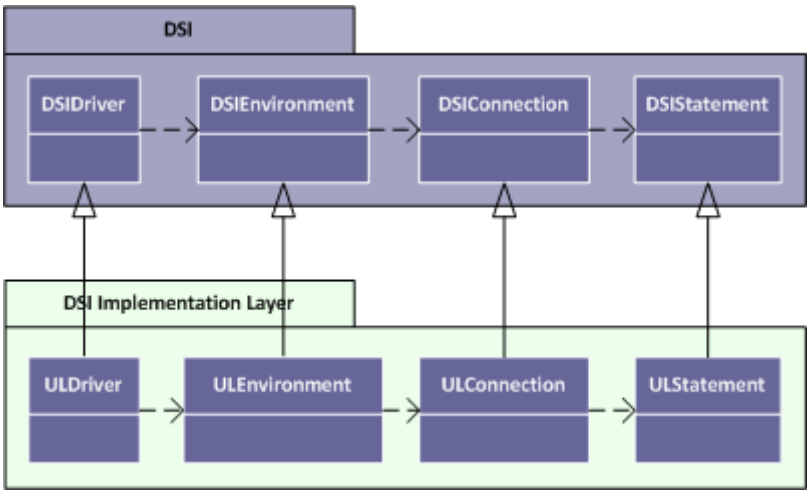


Figure 1 - Core Component Implementation

The DSIStatement implementation is responsible for creating a DSIDataEngine object which in turn creates IQueryExecutor objects to execute queries and hold results (IResults), and DSIMetadataSource objects to return metadata information. This is summarized in Figure 2:

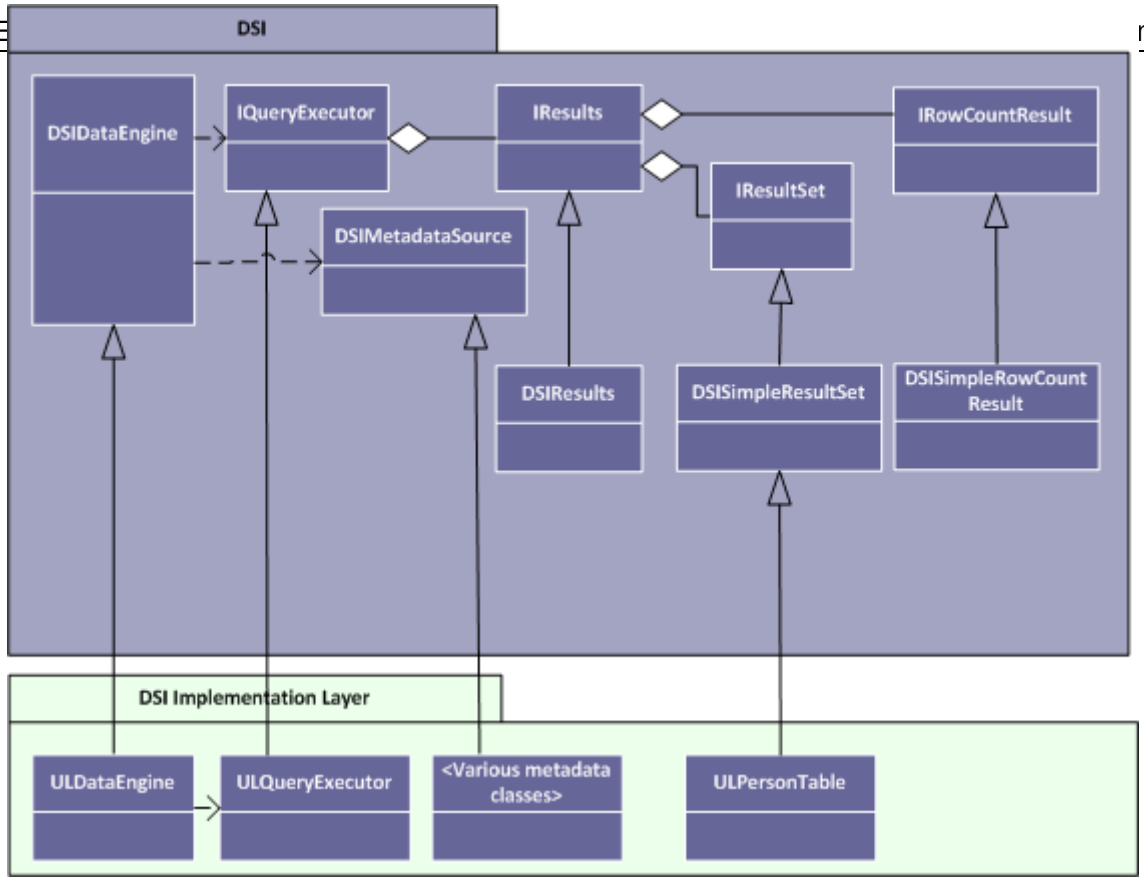


Figure 2 - DataEngine Implementation

The final key part of the DSI implementation is to create the framework necessary to retrieve both data and metadata. A summary of this framework and the components implemented by the sample are shown in Figure 3:

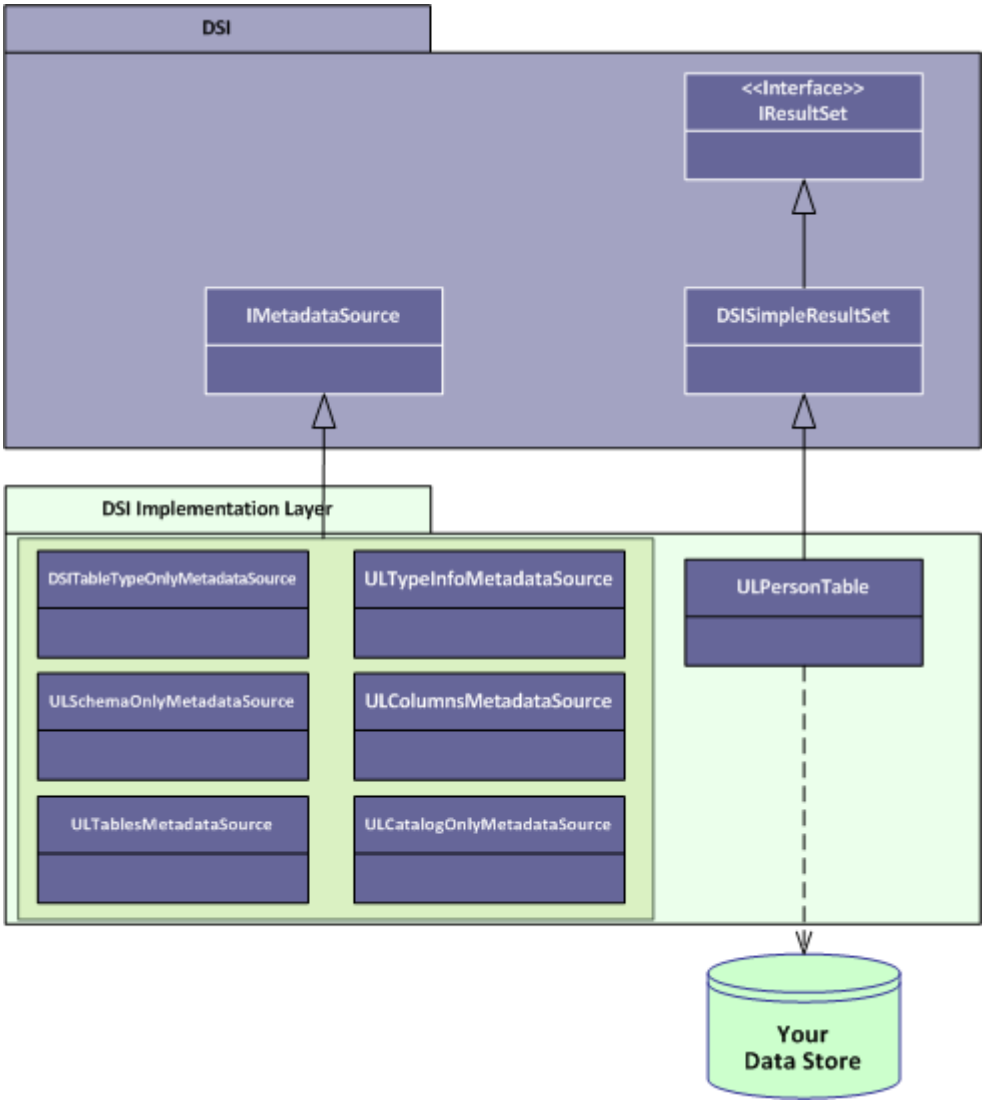


Figure 3: Design pattern for a DSI implementation.

The IResultSet interface is responsible for retrieving column data and maintaining a cursor across result rows.

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

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

Overview

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

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

In the JavaUltraLight 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 JavaUltraLight 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 first ODBC driver.

Install SimbaEngine

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

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

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 JavaUltraLight example driver

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

- The native C++ component (JavaUltraLightJNIDSI) 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 (JavaUltraLightDSII) 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_60`.
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_60\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 (UltraLightJNIDSI)

1. Launch Microsoft Visual Studio.
2. Click **File** > **Open** > **Project/Solution**.
3. Navigate to `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\JavaUltraLight\Source\JavaUltraLightJNIDSI` and then open the `UltraLightJNIDSI_VS201x.sln` file. This solution file contains the UltraLightJNIDSI, 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" and then click **Close**.
5. Click **Build** > **Build Solution** or press F7 to build the driver.
   This will build the debug version of the driver and place it in the following location for 32-bit drivers:

   ```
   [INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\source\JavaUltraLight\Bin\Win32\Debug
   ```

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

   ```
   [INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\JavaUltraLight\Bin\x64\Debug
   ```

Build the Java component (JavaUltraLightDSII)

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 JavaUltraLight 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 JavaUltraLight 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\JavaUltraLight\Source\JavaUltraLightDSII`. Click **OK** to the close the dialog.
3. Select only the JavaUltraLight project in the list of projects on the **Import dialog** and click **Finish**.
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: C:\Simba Technologies\SimbaEngineSDK\10.1\DataAccessComponents. Click **OK** to close the dialog, and the click **OK** to close the **Variable Selection** dialog.
5. A message is displayed, indicating that the classpath variables have changed and that a full rebuild is recommended. Click **No**.
6. Click **OK** to close the **Variable Entry** window and click **OK** to close the **Properties** window.
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 JavaUltraLight.
10. On the "Main" tab, in the **Buildfile** section, click **Browse Workspace**.
11. In the Choose Location window, click **JavaUltraLightBuilderODBC.xml** and then click **OK**.
12. Click **Apply**.
13. Still in the Ant build configuration window, 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 JavaUltralight driver using Apache Ant and it will place it in the location: `[INSTALL_DIRECTORY]\Examples\Source\JavaUltralight\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\JavaUltraLight\Source`.
2. For 32-bit Windows, open `SetupMyJavaUltraLightDSII-32on32.reg`.

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

   For a 64-bit ODBC driver on 64-bit Windows, open `SetupMyJavaUltraLightDSII-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 -Djava.class.path path value for JavaUltraLight.jar points to the correct location.
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 indicates that the keys and values have been successfully added to the registry.

Examine the registry keys for SimbaEngine

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

- **ODBC Data Sources** - lists each DSN/driver pair
- **JavaUltraLightDSII** - 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
- **JavaUltraLightDSIIDriver** - 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
   ```

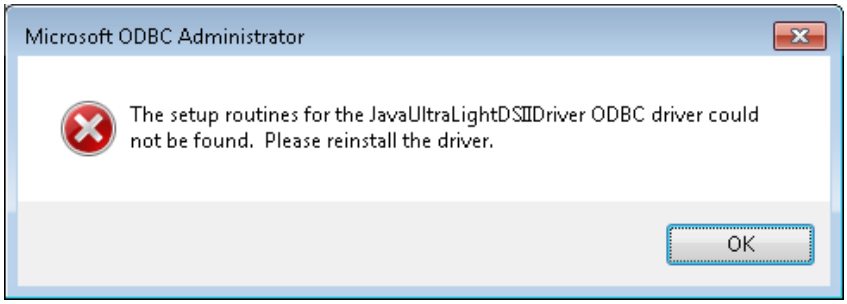
   For 32-bit drivers on 64-bit Windows:

   ```
   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 33 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 JavaUltraLight 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 'JavaUltraLightDSII'".

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\JavaUltraLight` directory and paste it to the same location. This will create a new directory called "`JavaUltraLight - 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 `JavaUltraLightJNIDSI` directory to `<YourProjectName>JNIDSI`.
4. Rename the `JavaUltraLightDSII` directory to `<YourProjectName>DSII`.
5. Go to the `<YourProjectName>JNIDSI` directory and then right-click the UltraLightJNIDSI_VS201x.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 **UltraLightJNIDSI_VS201x** solution to **<YourProjectName>JNIDSI_VS201x**.
9. Rename the **UltraLightJNIDSI** 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\CommonSource` directory and then rename the JavaUltraLightBuilder.xml file to `<YourProjectName>Builder.xml`.
13. In Windows Explorer, go to the `<YourProjectName>DSII\ODBC` directory and then rename the JavaUltraLightBuilderODBC.xml file to `<YourProjectName>BuilderODBC.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 “JavaUltraLight” 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.
14. Open the .project file in a text editor and replace the “JavaUltraLight” 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: Update full Java driver name.          (UltraLightJNIDSI.cpp)
TODO #2: Find or create the Java Virtual Machine.  (UltraLightJNIDSI.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 JavaUltraLight 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\ODBC\<YourProjectName>DSII`. Then click **Finish**.
2. To see your new Java project in the Project Explorer window, click Window > Show View > Project Explorer.
3. Select only the `<YourProjectName>` project in the list of projects on the **Import dialog** and click **Finish**.
4. Click **Run >External Tools > External Tools Configurations**.
5. In the External Tools Configurations window, double-click **Ant Build**.
   A setup page for the new Ant build configuration is displayed.
6. In the **Name** field, type `<YourProjectName>`.

7. On the "Main" tab, in the Buildfile section, click **Browse Workspace**.
8. In the **Choose Location** window, click `<YourProjectName>`BuilderODBC.xml and then click **OK**.
9. 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 name.`	(ULDriver.java)
`TODO #2: Set the driver properties.`	(ULDriver.java)
`TODO #3: Set the connection properties.`	(ULConnection.java)
`TODO #4: Set the driver-wide logging details.`	(ULDriver.java)
`TODO #5: Set the connection-wide logging details.`	(ULConnection.java)
`TODO #6: Check connection settings.`	(ULConnection.java)
`TODO #7: Establish a connection to your data store.`	(ULConnection.java)
`TODO #8: Create and return your Metadata Sources.`	(ULDataEngine.java)
`TODO #9: Prepare a Query.`	(ULDataEngine.java)
`TODO #10: Implement a Query Executor.`	(ULQueryExecutor.java)
`TODO #11: Provide parameter information.`	(ULQueryExecutor.java)
`TODO #12: Implement Query Execution.`	(ULQueryExecutor.java)
`TODO #13: Implement your Result Set.`	(ULPersonTable.java)
`TODO #14: Register your error messages for handling by DSIMessageSource`	(ULDriver.java)
`TODO #15: Set the vendor name, which will be prepended to error messages.`	(ULDriver.java)
`TODO #16: Update the component name.`	(UltraLight.java)
`TODO #17: Assign a unique component ID value to the messages.`	(UltraLight.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\<YourProjectName>\Source`.

2. For 32-bit Windows, open `SetupMyJavaUltraLightDSII-32on32.reg`.

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

   For a 64-bit ODBC driver on 64-bit Windows, open `SetupMyJavaUltraLightDSII-64on64.reg`.
3. In the file, replace all instances of JavaUltraLight with the name of your driver.
4. 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`.
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 `"MyJavaUltraLightDSII"="MyJavaUltraLightDSIIDriver"` 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\MyJavaUltraLightDSII]` 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 `"MyJavaUltraLightDSIIDriver"="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\MyJavaUltraLightDSIIDriver]` 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 "`JavaUltraLight`" 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

Testing the native (C++) project

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. Attach Visual Studio to the ODBC Test process. To do this, go to Microsoft Visual Studio and then click **Debug > Attach to Process**.
3. In the Attach to Process window, ensure that the **Attach to** field is set to "Native Code". In the list of available processes, select the ODBC Test process and then click **Attach**. The process name will be either `odbc32.exe` or `odbct32w.exe`.
4. Add a breakpoint to `Simba::JNIDSI::SetConfigurationBranding()` in `Main_Windows.cpp`. This code runs as soon as the Driver Manager loads the ODBC driver.
5. In the ODBC Test tool, select **Conn > Full Connect**.
   The Full Connect window opens.
6. Select your Data Source from the list of data sources and 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.
7. You should hit the breakpoint you created and focus should switch to Visual Studio.
8. To continue running the program, select **Debug > Continue**.
   The focus returns to the ODBC Test window.
9. When the tool connects to the data source, you will see the message "Successfully connected to DSN '<YourProjectName>DSII'".

Debugging your Driver

During the development of your driver, it may be necessary for you to trace through your driver during execution to locate problems. Most Java applications will either have a shell script/batch file to launch the application or have a configuration file where JVM options can be added.

You will need to add the following JVM options to enable debugging with the Eclipse IDE:

- `–Xdebug`
- `-Xrunjdwp:transport=dt_socket,address=localhost:8000,suspend=n,server=y`

These need to be added to the **JNIConfig** registry entry in

`HKEY_LOCAL_MACHINE/SOFTWARE/Simba/JavaUltraLight/Driver`

Each value needs to be separated by a pipe "|" character. For example:

```
-Djava.class.path=C:\Simba
Technologies\SimbaEngineSDK\10.1\Examples\Source\JavaUltraLight\Lib\JavaUltra
Light.jar|-Xdebug|-
Xrunjdwp:transport=dt_socket,address=localhost:8000,suspend=n,server=y
```

Once you have added these options for your application, launch your application. You will now be able to attach the Eclipse debugger to the running application and debug your driver.

If you need to debug the initialization of your driver, set the **suspend** parameter to **y**. A good breakpoint to start with is inside the `ULDriver` constructor. Launch your JDBC-enabled application. The JVM will suspend execution until a debugger has been attached.

Please see the Eclipse documentation for instructions on debugging Remote Java Applications.

By the end of this day, you should have built and tested, unchanged, the example driver shipped with SimbaEngine to make sure that your installation worked properly and that your development system is properly set up. Also, you should have created, built and tested a copy of the JavaUltraLight Driver example that you will change to work with your own data store.

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 on Linux.

Install SimbaEngine

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_9.2.0.1000.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 [INSTALL_DIRECTORY].
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 JavaUltraLight example driver

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

- The native C++ component (UltraLightJNIDSI) 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 (JavaUltraLight) is the DSII.

Build the native C++ component (UltraLightJNIDSI)

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

To build the SimbaEngine UltraLight 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/jdk1.7.0_60/
   ```

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=[INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/DataAccessCompone
   nts
   ```

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

   ```
   export
   SIMBAENGINE_THIRDPARTY_DIR=[INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/DataAcc
   essComponents/ThirdParty
   ```

6. In the commands from the previous two steps above, replace [INSTALL_DIRECTORY] with the directory where you installed the SimbaEngine files.
7. Type the following command to change to the directory that contains the makefile:

   ```
   cd
   [INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/Examples/Source/JavaUltraLight/Ma
   kefiles
   ```

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

   ```
   make –f ./UltraLightJNIDSI.mak debug
   ```

   Optionally, other options can be specified on the command line. If you are using the Xcode5 package, you must specify `COMPILER=Xcode5` in the make command or as an environment

variable. For more information about the options and build configurations, refer to the SimbaEngine Developer Guide.

Build the Java component (JavaUltraLight)

1. Launch the Eclipse IDE.
   Note: You can download the Eclipse IDE for Java Developers from www.eclipse.org/downloads.
2. Import the JavaUltraLight 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\JavaUltraLight\Source\JavaUltraLightDSII\ODBC`. Then click **Finish**.
3. Select only the JavaUltraLight project in the list of projects on the **Import dialog** and click **Finish**.
4. Select the JavaUltraLight project in the Package Explorer.
5. Click **Project > Properties > Java Build Path > Libraries**. Click **Add Variable** and then click on the **Configure Variables on the New Variable Classpath Entry** dialog. Click **New** on the **Preferences** dialog, enter **SIMBAENGINE_DIR** in the **Name** field, and the path in the **Path** field. For example, it might be something like this: `[INSTALL_DIRECTORY]`\SimbaEngineSDK\10.1\DataAccessComponents. Click **OK** to close the **Preferences** dialog and click **OK** to close the **Configure Variables on the New Variable Classpath Entry** dialog.
6. If a message is displayed, saying that the classpath variables have changed and that a full rebuild is recommended. Click **No**.
7. Click **OK** to close the properties windows.
8. Click **Run >External Tools > External Tools Configurations**.
9. In the External Tools Configurations window, double-click Ant Build.
   A setup page for the new Ant build configuration is displayed.
10. In the Name field, type JavaUltraLight.
11. On the "Main" tab, in the Buildfile section, click **Browse Workspace**.
12. In the Choose Location window, click JavaUltraLightBuilderODBC.xml and then click **OK**.
13. Click **Apply**.
14. Still in the Ant build configuration window, switch to the **Environment** tab. Add an environment variable called SIMBAENGINE_DIR with the value `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\DataAccessComponents`.
15. Click **Apply**.
16. Click **Run**. This will build the JavaUltraLight driver using Apache Ant and it will place it in the location:
    `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.1\Examples\Source\JavaUltraLight\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:

```
[INSTALL_DIRECTORY]/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 [INSTALL_DIRECTORY]/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]
   JavaUltraLightDSII=JavaUltraLightDSIIDriver
   ```
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.
   ```
   [JavaUltraLightDSII]
   Description=Sample 64-bit SimbaEngine JavaUltraLight DSII
   ```

```
Driver=[INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/Examples/Source/JavaUltraL
ight/Bin/Linux_x8664/libUltraLightJNIDSI_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
   [INSTALL_DIRECTORY]/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 the 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]
   JavaUltraLightDSIIDriver=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.

   ```
   [JavaUltraLightDSIIDriver]
   Driver=[INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/Examples/Source/JavaUltraL
   ight/Bin/Linux_x8664/libUltraLightJNIDSI_debug.so
   ```

Configure the Simba JavaUltraLight ODBC Driver

1. To see if the `.simba.javaultralight.ini` file already exists in your home directory, type the following command:
   `ls -al ~ | grep .simba.javaultralight.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
   [INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/Documentation/Setup/.simba.javault
   ralight.ini ~/.simba.javaultralight.ini
   ```
3. Open the `~/.simba.javaultralight.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=/usr/lib/libiodbcinst.so` (notice the 'i' after the lib)
   For unixODBC this would be:
   `ODBCInstLib=/usr/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 Developer Guide.

Enable Debugging

To enable debugging for the JavaUltraLight sample driver:

1. In the .simba.javaultralight.ini file that you edited in the section *Configure the Simba JavaUltraLight ODBC Driver* on page 19, update the JNIConfig line to indicate the following:

   ```
   JNIConfig=-Djava.class.path=[INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/
   Examples/Source/JavaUltralight/Lib/JavaUltraLight.jar
   ```

   OR

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

   ```
   JNIConfig=-Djava.class.path=[INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/
   Examples/Source/JavaUltraLight/Lib/JavaUltraLight.jar|-Xdebug|-Xrunjdwp
   :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
```

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/DataAcc
essComponents/ThirdParty/icu/53.1/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 Developer Guide.

  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 JavaUltraLightDSII DSN.
  4. To connect to your data source, type the following command:

     ```
     DSN=JavaUltraLightDSII
     ```

     A prompt that says “SQL>” appears.
  5. Type a SQL command to query your database. For example, `SELECT * FROM PRODUCT`. A simple result set should be returned.

  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 JavaUltraLight 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
   [INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/Examples/Source/JavaUltraLight
   [INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/Examples/MyJavaUltraLight
   ```

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 `UltraLightJNIDSI.mak` file in it. For example, you could type `mv UltraLightJNIDSI.mak MyUltraLightJNIDSI.mak`.

3. Then, rename the `.depend` file that is located in the `Makedepend` directory.

4. Open your new directory then open the `Source/JavaUltraLightJNIDSI` directory. Open the `Makefile` file and replace the "`UltraLightJNIDSI`" 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:
`[INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/Examples/Source/MyJavaUltraLight/Makefiles`

6. To run the makefile for the debug target, type the following command:

```
make –f MyUltraLightJNIDSI.mak 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]
MyJavaUltraLightDSII=MyJavaUltraLightDSIIDriver
```

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

```
[MyJavaUltraLightDSII]
Description=My SimbaEngine DSII
Driver=[INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/Examples/Source/MyJavaUltraLight/Bin/Linux_x8664/libMyUltraLightJNIDSI_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]
MyJavaUltraLightDSIIDriver=Installed
```

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

```
[MyJavaUltraLightDSIIDriver]
Driver=[INSTALL_DIRECTORY]/SimbaEngineSDK/10.1/Examples/Source/MyJavaUltraLight/Bin/Linux_x8664/libMyUltraLightJNIDSI_debug.so
```

7. Copy the `.simba.javaultralight.ini` file in your home directory so it has the name `.simba.myjavaultralight.ini`, and then update settings in `.simba.myjavaultralight.ini` for your own ODBC driver as described in *Configure the Simba JavaUltraLight ODBC Driver* on page 19.

8. To enable debugging, in the `.simba.myjavaultralight.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 MyJavaUltraLightDSII DSN.
4. To connect to your data source, type the following command:

   ```
   DSN=MyJavaUltraLightDSII
   ```

   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 JavaUltraLight 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 JavaUltraLight 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: Update full Java driver name.`	(UltraLightJNIDSI.cpp)
`TODO #2: Find or create the Java Virtual Machine.`	(UltraLightJNIDSI.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 `UltraLightJNIDSI.cpp` in the `UltraLightJNIDSI` 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()`. The name returned by this method must match the fully-qualified name of the Java driver class name in ULDriver.java (e.g. com.simba.ultralight.core.ULDriver).

Set the driver name

`TODO #1: Set the driver name.`	(ULDriver.java)

Set the constant DRIVER_NAME to the name of your driver (usually the same name you used to replace “JavaUltraLight” in the section “Set up a new project to build your own ODBC driver” from Day One).

Set the driver properties

`TODO #2: Set the driver properties.`	(ULDriver.java)
`TODO #3: Set the connection properties.`	(ULConnection.java)

1. In Eclipse, in your project, go to the TODO #2 message in the `ULDriver.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.
3. Go to the TODO #3 message in the `ULConnection.java` file. Look at `setProperty()`where you will set up the general properties for your connection.
4. Adjust the connection properties as required by your driver.

Set the logging details

`TODO #4: Set the driver-wide logging details.`	(ULDriver.java)
`TODO #5: Set the connection-wide logging details.`	(ULConnection.java)

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

Note: By default, the SimbaEngine JavaUltraLight 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 Developer Guide.

Check the connection settings

`TODO #6: Check Connection Settings.`	(ULConnection.java)

When the Simba JDBC layer is given a connection URL from a JDBC-enabled application, the Simba JDBC layer parses the connection string into key-value pairs. Then, the entries in the

connection string and the DSN are sent to `updateConnectionSettings()`function which is responsible for verifying that all of the required, and any optional, connection settings are present.

When a connection occurs, a connection URL is passed to the JDBC driver. As an example, take the connection string “jdbc:simba://User=user;Password=pass”. This connection string is broken down into key-value pairs and stored in a ConnSettingRequestMap, in this case that map would contain two entries: {“User”, “user”} and {“Password”, “pass”}. This map is then passed down to the DSII.

1. Go to the TODO #6 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 #7: Establish A Connection.                    (ULConnection.java)
```

Once `ULConnection.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 `ULConnection`'s `connect()` method 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()`.

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 (SCHEMA_ONLY)
- SQLTables (TABLE_TYPE_ONLY)
- SQLTables
- SQLColumns

Create and return metadata sources

```
TODO #8: Create and return your Metadata Sources.    (ULDataEngine.java)
```

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

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

Handle TYPE_INFO

The underlying ODBC catalog function SQLGetTypeInfo is handled as follows:

1. When called with `TYPE_INFO`, `ULDataEngine.makeNewMetadataSource()` will return an instance of `ULTypeInfoMetadataSource()`.
2. The SimbaEngine JavaUltraLight Driver example exposes support for the following types:

SQL_BIT	SQL_CHAR	SQL_DOUBLE
SQL_INTEGER	SQL_LONGVARBINARY	SQL_LONG_VARCHAR
SQL_NUMERIC	SQL_REAL	SQL_SMALLINT
SQL_TINYINT	SQL_TYPE_DATE	SQL_TYPE_TIME
SQL_TYPE_TIMESTAMP	SQL_VARBINARY	SQL_VARCHAR
SQL_WCHAR	SQL_WLONGVARCHAR	SQL_WVARCHAR

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

Handle the other MetadataSources

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

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

- ULSchemaOnlyMetadataSource: returns a list of all schemas. The sample implementation returns one row of information with one column containing the name of a fake schema. This demonstrates how to return a schema name.
- ULTablesMetadataSource: returns metadata about all of the tables in the data source. The sample hard codes and returns information for the hard coded person table to demonstrate how to return table metadata.
- ULColumnsMetadataSource: returns metadata for the columns in the data source. The sample hard codes and returns information for the three columns in the person table consisting of the name column, an integer column, and a numeric column.

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

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

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

Query Preparation

The first step in obtaining data from your data store is to prepare a query.

```
TODO #9: Prepare a query (ULDataEngine.java).
```

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

For demonstration purposes, the default implementation of `ULDataEngine.prepare()` performs a very simple preparation by searching for the substrings "select" and "?" in the query and sets up a collection of statement types to be passed to a `ULQueryExecutor` for processing.

If "select" is found, then it is assumed that the caller wants to search for rows of data and a result set is therefore returned. If "select" is not found, then it is assumed that the caller wants to retrieve the number of rows and so a row count is therefore returned. If "?" is present, then the statement is assumed to be parameterized and therefore `ULQueryExecutor`'s constructor will populate parameters as described below.

The method also checks the query for the string “{multipleresultset” and if found will create and return a row count. If the “multipleresultset” string does not follow the “{“ character, then two result sets are returned.

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

If the query can be prepared, a new instance of your `IQueryExecutor` will be returned.

Query Execution

After a query has been prepared, a query is executed.

```
TODO #10: Implement a QueryExecutor (ULQueryExecutor.java)
```

The ULQueryExecutor object returned by the `ULDataEngine.prepare()` method is an implementation of IQueryExecutor which, as the name suggests, executes a query. The implementation of ULQueryExecutor simply checks each statement type passed in via the collection of statement types. If a statement is a select statement then the constructor creates a simple result set consisting of people’s names and adds it to `m_Results`. Otherwise, it creates and adds a row count.

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

```
TODO #11: Provide parameter information (ULQueryExecutor.java).
```

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

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

```
TODO #12: Implement Query Execution (ULQueryExecutor.cs).
```

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

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

After this method exits, the calling framework will then obtain the results by invoking the `ULQueryExecutor.getResults()` method.

Query Results

After a query has been executed, the query results are returned in an implementation of the `IResultSet` interface. The `DSISimpleResultSet` class provides a partial implementation of the interface to simplify the task of implementing a basic forward-only, read-only result set.

```
TODO #13: Implement your Result Set. (ULPersonTable.java)
```

The final step in returning data is to implement `DSISimpleResultSet`. The sample contains an implementation called `ULPersonTable` which returns a hardcoded set of people's names. In general, your "table" class can represent the results of a query that may involve more than a single table but for simplicity, this tutorial assumes a query involving a single table.

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

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

In the example, `ULPersonTable.doMoveToNextRow()` (which is invoked by `moveToNextRow()`) simply returns a boolean indicating whether or not the driver is on the last row of data, so this should be replaced in your implementation with code that delegates this to the data source.

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

The `doCloseCursor()`method should be implemented because it is called by SimbaEngine to indicate that data retrieval has completed and that you may now perform any tasks related to closing any associated result set in your data store.

Similarly, `hasMoreRows()` should be implemented to indicate if there are any more rows to fetch after the current row.

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 Developer Guide for more details.

```
TODO #14: Register your error messages.                    (ULDriver.java)
```

For the purpose of prototyping, this TODO is purely informational. By default, the driver's messages.properties file resides in the same package as the ULDriver 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.

```
TODO #15: Set the vendor name.                    (ULDriver.java)
```

All error messages returned by the driver begin with the vendor name. The default vendor name is "Simba". Simply uncomment the call to the `setVendorName()` method and replace "`vendorName`" with an appropriate name for your organization. This will rebrand your converted JavaUltraLight driver for your organization.

```
TODO #16: Update the component name.              (UltraLight.java)
```

All error messages returned by your DSII contain the component name. Simply change "`UltraLightDSII`" to a name relating to your driver. This will rebrand your converted JavaUltraLight Driver for your organization.

```
TODO #17: Assign a unique component ID.           (UltraLight.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.

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 UltraLight 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 UltraLight 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 32 for details.

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

You are now done with all of the TODO's in the project. You have created your own, custom ODBC driver using SimbaEngine by modifying and customizing the JavaUltraLight 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 is one of the few applications 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:

- **JavaUltraLightDSII** which must include the following string values:
  - **Driver:** `[INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release\UltraLightJNIDSI.dll`
  - **Description:** `Sample 32-bit SimbaEngine JavaUltraLight DSII`

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

- **ODBC Data Sources** which must include the following string value:
  - **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** and its **Description** to help you clearly identify each registry key. The keys that are relevant to the C# examples discussed in this document are:

- **JavaUltraLightDSIIDriver** which includes the following key names and values:
  - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightJNIDSI.dll`
  - **Description:** `Sample 32-bit SimbaEngine JavaUltraLight DSII`

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

- **JavaUltraLightDSII** which must include the following string values:
  - **Driver:** `[INSTALL_DIRECTORY]\Examples\Builds\Bin\Win32\Release\UltraLightJNIDSI.dll`
  - **Description:** `Sample 32-bit SimbaEngine JavaUltraLight DSII`

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

- **ODBC Data Sources** which must include the following string values:
  - **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** and a **Description** to help you clearly identify each registry key. The keys that are relevant to the examples discussed in this document are:

- **JavaUltraLightDSIIDriver** which includes the following key names and values:
  - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\Win32\Release\UltraLightJNIDSI.dll`
  - **Description:** `Sample 32-bit SimbaEngine JavaUltraLight DSII`

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

- **JavaUltraLightDSII** which must include the following string values:
  - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\x64\Release\UltraLightJNIDSI.dll`
  - **Description:** `Sample 64-bit SimbaEngine JavaUltraLight DSII`

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

- **ODBC Data Sources** which must include the following string values:
  - **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** and a **Description** to help you clearly identify each registry key. The keys that are relevant to the examples discussed in this document are:

- **JavaUltraLightDSIIDriver** which includes the following key names and values:
  - **Driver:** `[INSTALL_DIRECTORY]\Examples\Bin\x64\Release\UltraLightJNIDSI.dll`
  - **Description:** `Sample 64-bit SimbaEngine JavaUltraLight DSII`

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 value:
  - **JavaUltraLightDSIIDriver:** `Installed`

Appendix C: Data Retrieval

In the Data Store Interface (DSI), the following two methods perform the task of retrieving data from your data store:

1. Each `IMetadataSource` implementation of `getMetadata()`
2. `ULPersonTable'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 `ULPersonTable`, 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 `ULPersonTable's initializeColumns()` as `Types.INTEGER`, then when `ULPersonTable'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), SimbaEngine 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 ULPersonTable'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 `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: 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.

IMPORTANT: The information in this section only applies if you are using 32-Bit Windows. If you are using 64-bit Windows (with either 32-bit or 64-bit applications), the file paths must be configured appropriately. Please see Appendix B: Windows Registry 32-Bit vs. 64-Bit on page 33 for details.

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

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

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

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

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

1. Build UltraLightJNIDSI using the debug (or release) server configuration:
   ```
   BUILDSERVER=exe make –f UltraLightJNIDSI.mak debug
   ```
2. Configure the server as required in the other sections of the .simbaserver.javaultralight.ini file.

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

Third Party Licenses

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

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

All rights reserved.

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

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

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

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

OpenSSL License

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

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

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

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

3. All advertising materials mentioning features or use of this software must display the following acknowledgment:

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

4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.

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 NOINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

Stringencoders License

Copyright 2005, 2006, 2007