SimbaEngine SDK 9.3

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

Last Revised: June 2014

Simba Technologies Inc.
Table of Contents

Build an ODBC Driver for SQL-Based Data Sources in Five Days ............................................. 1
  About the SimbaEngine SDK ................................................................................................. 1
  About the UltraLight sample driver ...................................................................................... 2
  Overview ................................................................................................................................. 5

Day One – Windows Instructions .................................................................................................. 6
  Install the SimbaEngine SDK ................................................................................................. 6
  Build the UltraLight example driver ....................................................................................... 6
  Examine the registry keys added by the SimbaEngine SDK installer ......................................... 6
  View the data source in the ODBC Data Source Administrator ............................................... 7
  Test the data source .................................................................................................................. 8
  Set up a new project to build your own ODBC driver ............................................................... 8
  Build your new driver ............................................................................................................... 9
  Update the registry .................................................................................................................... 10
  View your new data source in the ODBC Data Source Administrator ...................................... 11
  Test your new data source ....................................................................................................... 11
  Summary – Day One .................................................................................................................. 12

Day One – Linux Instructions ...................................................................................................... 13
  Install the SimbaEngine SDK ................................................................................................. 13
  Build the UltraLight sample driver ....................................................................................... 13
  Configure the ODBC data source and ODBC driver ............................................................... 14
  Test the data source .................................................................................................................. 17
  Build your new ODBC driver ................................................................................................... 17
  Configure an ODBC data source and ODBC driver ............................................................... 18
  Test your new data source ....................................................................................................... 19
  Summary – Day One .................................................................................................................. 20

Day Two ....................................................................................................................................... 20
  Construct a driver singleton ...................................................................................................... 20
  Set the driver properties .......................................................................................................... 21
  Set the logging details ............................................................................................................... 21
  Check the connection settings ................................................................................................. 22
  Customize the DriverPrompt Dialog ....................................................................................... 23
  Establish a connection .............................................................................................................. 24

Day Three .................................................................................................................................... 24
  Create and return metadata sources ........................................................................................ 24

Day Four ...................................................................................................................................... 26
  Prepare and execute a query ...................................................................................................... 26

Day Five ...................................................................................................................................... 28
  Configure error messages ........................................................................................................ 28
  Finishing touches ...................................................................................................................... 29

Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit ......................... 30
Appendix B: Windows Registry 32-Bit vs. 64-Bit ......................................................................... 31
  32-Bit Drivers on 64-Bit Windows ............................................................................................ 31
  64-Bit Drivers on 64-Bit Windows ........................................................................................... 32
Appendix C: Data Retrieval ......................................................................................................... 34
  SqlData types .......................................................................................................................... 35
Build an ODBC Driver for SQL-Based Data Sources in Five Days

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

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

About the SimbaEngine SDK

The SimbaEngine SDK is a complete implementation of the ODBC specification, which provides a standard interface to which any ODBC enabled application can connect. The libraries of the SimbaEngine SDK 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 the SimbaEngine SDK is available on the Simba website at http://www.simba.com/odbc-sdk-documents.htm.

You use the SimbaEngine SDK 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 (DSI) that connects directly to your data source. Then, you create the executable by linking libraries from SimbaEngine SDK 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 SDK take responsibility for meeting the data access standards while your custom DSI implementation takes responsibility for accessing your data store and translating it to the DSI API.
About the UltraLight sample driver

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

The UltraLight driver helps you to prototype a DSI implementation for your own SQL-based data store. 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 DSI Driver class which is responsible for constructing a DSI Environment. This in turn is used to construct a connection object (DSI Connection implementation) which can then be used for constructing statements (DSI Statement implementations). This is summarized in Figure 1.

![Figure 1 – Core Component Implementation](image)

The DSI Statement 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:
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 2 – DataEngine Implementation**
Figure 3 - Design pattern for a DSI implementation.

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

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

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

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

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

In the UltraLight driver, the areas of the code that you need to change are marked with “TODO” messages along with a short explanatory message. Most of the areas of the code that you need to modify are for productization such as naming the driver, setting the properties that configure the driver, and naming the XML error file and log files. The other areas of the code that you will modify are related to getting the data and metadata from your data store. Since the UltraLight driver already has the classes and code to do this against its example data store (hard coded data), all you have to do is modify the existing code to make your driver work against your own data store.
Day One – Windows Instructions

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

Install the SimbaEngine SDK

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

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

Build the UltraLight example driver

1. Launch Microsoft Visual Studio.
2. Click File > Open > Project/Solution.
3. Navigate to [INSTALLDIR]\SimbaEngineSDK\9.3\Examples\Source\UltraLight\Source and then open the UltraLight_vs201x.sln file.
   The default [INSTALLDIR] is C:\Simba Technologies.
4. Click Build > Configuration Manager and make sure that the active solution configuration is “Debug” and then click Close.
5. Click Build > Build Solution or press F7 to build the driver. This will build the debug version of the driver and place it in the following location for 32-bit drivers:

   [INSTALLDIR]\SimbaEngineSDK\9.3\Examples\Source\UltraLight\Bin\Win32\Debug

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

   [INSTALLDIR]\SimbaEngineSDK\9.3\Examples\Source\UltraLight\Bin\x64\Debug

Examine the registry keys added by the SimbaEngine SDK installer

The SimbaEngine SDK installer automatically added or updated the following registry keys that define Data Source Names (DSNs) and driver locations:

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

To view the registry keys, do the following:

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

   For 32-bit drivers on 32-bit Windows and 64-bit drivers on 64-bit Windows:
   ```
   HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBC.INI
   ```

   For 32-bit drivers on 64-bit Windows:
   ```
   HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBC.INI
   ```

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

   For 32-bit drivers on 32-bit Windows and 64-bit drivers on 64-bit Windows:
   ```
   HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI
   ```

   For 32-bit drivers on 64-bit Windows:
   ```
   HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBCINST.INI
   ```

Your custom driver installer will 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 31 for more information.

**View the data source in the ODBC Data Source Administrator**

1. Run the Windows ODBC Data Source Administrator.

   For 32-bit drivers on 32-bit Windows and 64-bit drivers on 64-bit Windows, open the Control Panel, select Administrative Tools, and then select Data Sources (ODBC). If your Control Panel is set to view by category, then Administrative Tools is located under System and Security.

   For 32-bit drivers on 64-bit Windows, you must use the 32-bit ODBC Data Source Administrator. You cannot access the 32-bit ODBC Data Source Administrator from the start menu or control panel on 64-bit Windows. Only the 64-bit ODBC Data Source Administrator is accessible from the start menu or control panel. On 64-bit Windows, to launch the 32-bit ODBC Data Source Administrator you must run `C:\WINDOWS\SysWOW64\odbcad32.exe`. See Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 30 for details.

2. In the ODBC Data Source Administrator, click the **System DSN** tab.
3. Scroll through the list of System Data Sources, select UltraLightDSII and then click Configure. The Data Source Configuration window opens and displays the fields for the user ID, password, and language.

4. Now that you have looked at the configuration information for the driver, click Cancel to close the Data Source Configuration window.

Test the data source

To test the data source that we have created, you can use any ODBC application, such as, for example, Microsoft Excel, Microsoft Access or ODBCTest. In this section, we will use the ODBC Test tool, which is available in the Microsoft Data Access (MDAC) 2.8 Software Development Kit (SDK). To download the SDK, visit the following Microsoft Web site: [http://www.microsoft.com/downloads/details.aspx?FamilyID=5067faf8-0db4-429a-b502-de4329c8c850&displaylang=en](http://www.microsoft.com/downloads/details.aspx?FamilyID=5067faf8-0db4-429a-b502-de4329c8c850&displaylang=en)

1. Start the ODBC Test tool. By default, the ODBC Test application is installed in the following folder: C:\Program Files (x86)\Microsoft Data Access SDK 2.8\Tools\ Navigate to the folder that corresponds to your machine’s architecture (amd64, ia64 or x86) and then click 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 ‘UltraLightDSII’”.

Set up a new project to build your own ODBC driver

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

**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 the SDK, changes you make will be lost and there may be times, for debugging purposes, that you will need to see if the same error occurs using the sample drivers. If you have modified the sample drivers, this won’t be possible.

1. In your Windows Explorer window, copy the [INSTALLDIR]\SimbaEngineSDK\9.3\Examples\Source\UltraLight directory and paste...
it to the same location. This will create a new directory called "UltraLight - Copy". Rename the directory to something that is meaningful to you. This will be the top-level directory for your new project and DSI implementation files. For the rest of this tutorial, when you see <YourProjectName> in the instructions, replace this with the name you choose for this directory which is also the name of your project.

2. Open your new directory then open the Source directory and rename the UltraLight_vs201x.vcxproj file in it to <YourProjectName>.vcxproj file where you replace <YourProjectName> with the name of your project. This will be the project file for your new ODBC driver.

3. Rename the .sln file. This new <YourProjectName>.sln file is the solution file for your new ODBC driver.

4. Using a text editor, open the project file (.vcxproj) and replace every instance of "UltraLightDSII" in the source code with the name of your new ODBC driver. Then save and close the file.

5. Using a text editor, open the solution file (.sln) and replace every instance of "UltraLightDSII" in the source code with the name of your new ODBC driver. In addition, reference to the name of the project file must be updated to match the <YourProjectName>.vcxproj project file that you renamed. Then, save and close the file.

Build your new driver

1. Launch Microsoft Visual Studio.

2. Click File > Open > Project/Solution.

3. Navigate to [INSTALLDIR]\SimbaEngineSDK\9.3\Examples\Source\<YourProjectName>\Source and then open the <YourProjectName>.sln file.

4. Click Build > Configuration Manager and make sure that the active solution configuration is "Debug" 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 location: [INSTALLDIR]\SimbaEngineSDK\9.3\Examples\Source\<YourProjectName>\Bin\Win32\Debug.

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

<table>
<thead>
<tr>
<th>TODO #1</th>
<th>Construct driver singleton.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TODO #2</td>
<td>Set the driver properties.</td>
</tr>
<tr>
<td>TODO #3</td>
<td>Set the driver-wide logging details.</td>
</tr>
<tr>
<td>TODO #4</td>
<td>Set the connection-wide logging details.</td>
</tr>
<tr>
<td>TODO #5</td>
<td>Check Connection Settings.</td>
</tr>
<tr>
<td>TODO #6</td>
<td>Customize DriverPrompt Dialog.</td>
</tr>
<tr>
<td>TODO #7</td>
<td>Establish A Connection.</td>
</tr>
</tbody>
</table>
TODO #8: Create and return your Metadata Sources.
TODO #9: Prepare a Query.
TODO #10: Implement an IQueryExecutor.
TODO #11: Provide parameter information.
TODO #12: Implement Query Execution.
TODO #13: Implement your DSISimpleResultSet.
TODO #14: Register the ULMessages.xml file for handling by DSIMessageSource.
TODO #15: Set the vendor name, which will be prepended to error messages.

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

Update the registry

To update the registry keys, do the following:

1. In Microsoft Visual Studio, click File > Open > File and navigate to [INSTALLDIR]\SimbaEngineSDK\9.3\Examples\Source\MyUltraLight\Source.
2. For 32-bit Windows, open SetupMyUltraLightDSII-32on32.reg.
   For a 32-bit ODBC driver on 64-bit Windows, open SetupMyUltraLightDSII-32on64.reg.
   For a 64-bit ODBC driver on 64-bit Windows, open SetupMyUltraLightDSII-64on64.reg.
3. In the file, replace [INSTALLDIR] with the path to the installation directory. In the path, you must enter double backslashes. For example, by default, the samples are installed to “C:\Simba Technologies” so in that case, you would replace all instances of [INSTALLDIR] with C:\Simba Technologies.
4. Next, update the ODBC Data Sources section to add your new data source. Under the [HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources] section, change "MyUltraLightDSII"="MyUltraLightDSIIDriver" to the name of your new data source and new driver. For example, "<YourProjectName>DSII"="<YourProjectName>DSIIDriver"
5. Then, modify the data source definition for that data source. Change the line that says [HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\MyUltraLightDSII] so that it contains your new data source name. For example, [HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\<YourProjectName>DSII]
6. Beside the line that starts with "Driver"= change the driver name to that of your new ODBC driver. For example, "Driver"="<YourProjectName>DSIIDriver"
7. Update the ODBC Drivers section to add your new driver. Under the [HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers] section, change "MyUltraLightDSIIDriver"="Installed" to match the name of your new driver. For example, "<YourProjectName>DSIIDriver"="Installed"
8. Modify the driver definition for that driver. Change the line that says [HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\MyUltraLightDSIIDriver] so
that it contains your new driver name. For example,

```
[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\<YourProjectName>DSIIDriver]
```

9. Beside the lines that starts with “Driver” and “Setup”, update the path to the dll file for both.

10. Click **Edit > Find and Replace > Quick Replace**. Then, replace “UltraLight” in the whole file with the name of your new ODBC driver.

11. Click **Save** and then close the file.

12. In the Registry Editor (**regedit.exe**), click **File > Import**, navigate to the registry file that you just modified and then click **Open**.
   A message is displayed that says that the keys and values have been successfully added to the registry.

**View your new data source in the ODBC Data Source Administrator**

1. Run the Windows ODBC Data Source Administrator.

   For 32-bit drivers on 32-bit Windows and 64-bit drivers on 64-bit Windows, open the Control Panel, select Administrative Tools, and then select Data Sources (ODBC). If your Control Panel is set to view by category, then Administrative Tools is located under System and Security.

   For 32-bit drivers on 64-bit Windows, you must use the 32-bit ODBC Data Source Administrator. You cannot access the 32-bit ODBC Data Source Administrator from the start menu or control panel on 64-bit Windows. Only the 64-bit ODBC Data Source Administrator is accessible from the start menu or control panel. On 64-bit Windows, to launch the 32-bit ODBC Data Source Administrator you must run `C:\WINDOWS\SysWOW64\odbcad32.exe`. See Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 30 for details.

2. In the ODBC Data Source Administrator, click the **System DSN** tab.

3. Scroll through the list of System Data Sources, select `<YourProjectName>DSII` and then **click Configure**.
   The Data Source Configuration window opens and displays the data source name, description and the data directory.

4. Now that you have looked at the configuration information for your new driver, click **Cancel** to close the Data Source Configuration window.

**Test your new data source**

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

   Navigate to the folder that corresponds to your driver’s architecture (amd64, ia64 or x86) and then click `odbcite32.exe` to launch the ANSI version or click `odbc32w.exe` to launch the Unicode version. It is important to run the correct version of the ODBC Test tool for ANSI or Unicode and 32-bit or 64-bit.
2. 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, select the ODBC Test process and then click **Attach**. The process name will be either `odbc32.exe` or `odbct32w.exe`.

4. **Add a breakpoint in** `Main_Windows.cpp`, **on the function DSIDriverFactory()**. **This function runs as soon as the Driver Manager loads the ODBC driver.**

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

6. **Select your Data Source from the list of data sources and 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. **Enter Select * from ULResultSet in ODBC Test and click the button on the toolbar with the exclamation icon. Then click the button beside it. This will output a simple result set.**

**Summary – Day One**

At this point, you have completed the following tasks:

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

Install the SimbaEngine SDK

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

On Linux and UNIX platforms, the SimbaEngine SDK 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_Release_Linux-x86_9.3.0.1000.tar.gz

1. Open a command prompt.
2. Change to the directory where you want to install the SimbaEngine SDK. 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`

Build the UltraLight sample driver

On Linux and UNIX platforms, the sample drivers include makefiles instead of Visual Studio solution files. On these platforms, the process to build each of the sample drivers is similar. The exact process depends on whether you are using 32-bit or 64-bit Linux. To determine which version of Linux you are using, type `uname -m`.

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

1. Set the `SIMBAENGINE_DIR` environment variable by typing:
   ```bash
   export SIMBAENGINE_DIR=[INSTALLDIR]/SimbaEngineSDK/9.3/DataAccessComponents
   
   In the command above, replace [INSTALLDIR] with the directory where you installed the SimbaEngine SDK files.
   ```
2. Set the `SIMBAENGINE_THIRDPARTY_DIR` environment variable by typing:
   ```bash
   export SIMBAENGINE_THIRDPARTY_DIR=[INSTALLDIR]/SimbaEngineSDK/9.3/DataAccessComponents/ThirdParty
   
   In the command above, replace [INSTALLDIR] with the directory where you installed the SimbaEngine SDK files.
   ```
3. Change to the following directory:
   ```bash
   [INSTALLDIR]/SimbaEngineSDK/9.3/Examples/Source/UltraLight/Makefiles
   ```
4. Type `make -f UltraLight.mak debug` to run the makefile for the debug target. Note that, optionally, other options can be specified on the command line. For more information about the options and build configurations, refer to the SimbaEngine SDK Developer Guide.

**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:

```bash
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/9.3/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:

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

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

   ```bash
   -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:

   ```bash
   ```

   **Note:** the “.” before `odbc.ini` in `~/.odbc.ini` will cause the copied file to be hidden.

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. Replace every instance of `[INSTALLDIR]` with the installation location of the SimbaEngine SDK.
4. 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]
UltraLightDSII=UltraLightDSII

5. 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.

For 32-bit Linux, it might look something like this:

[UltraLightDSII]
Description=Sample 32-bit SimbaEngine UltraLight DSII
Driver=[INSTALLDIR]/SimbaEngineSDK/9.3/Examples/Source/UltraLight/Bin/Linux_x86/libUltraLight_debug.so

For 64-bit Linux, it might look something like this:

[UltraLightDSII]
Description=Sample 64-bit SimbaEngine UltraLight DSII
Driver=[INSTALLDIR]/SimbaEngineSDK/9.3/Examples/Source/UltraLight/Bin/Linux_x8664/libUltraLight_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:

   Note: the “.” before odbcinst.ini in ~/.odbcinst.ini will cause the copied file to be hidden.

2. Open the ~/.odbcinst.ini configuration file in a text editor.

3. Replace every instance of [INSTALLDIR] with the installation location of the SimbaEngine SDK.

4. 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]
UltraLightDSIIDriver=Installed

5. 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.

For 32-bit Linux, it might look something like this:

[UltraLightDSIIDriver]
Driver=\INSTALLDIR\SimbaEngineSDK\9.3\Examples\Source\UltraLight\Bin\Linux_x86\libUltraLight_debug.so

For 64-bit Linux, it might look something like this:

[UltraLightDSIIDriver]
Driver=\INSTALLDIR\SimbaEngineSDK\9.3\Examples\Source\UltraLight\Bin\Linux_x8664\libUltraLight_debug.so

Configure the Simba UltraLight ODBC Driver

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

   ls -al ~ | grep .simba.ultralight.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\9.3\Documentation\Setup\simba.ultralight.ini ~/.simba.ultralight.ini

3. Open the ~/.simba.ultralight.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_WCHAR_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/libodbcinst.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 SDK Developer Guide.

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:

```bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:[INSTALLDIR]/SimbaEngineSDK/9.3/DataAccessComponents/ThirdParty/icu-3.8.1/Linux_x86/lib
```

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

```bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:
[INSTALLDIR]/SimbaEngineSDK/9.3/DataAccessComponents/ThirdParty/icu-3.8.1/Linux_x8664/lib
```

- 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 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:

1. At the command prompt, type: `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 UltraLightDSII DSN.
4. To connect to your data source, type: `DSN=UltraLightDSII;UID=a;PWD=b`. A prompt that says “SQL>” appears.
5. Type a SQL command to query your database. For example, `SELECT * FROM ULResultSet`. This will output a simple result set.

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 UltraLight 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:

**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 UltraLight.mak file in it. For example, you could type `mv UltraLight.mak MyUltraLight.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 "UltraLight" 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/9.3/Examples/Source/MyUltraLight/Makefiles

6. Type `make -f MyUltraLight.mak debug` to run the makefile for the debug target.

**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]
   MyUltraLightDSII=MyUltraLightDSIIDriver

3. Make sure there is a section with a name that matches the data source name (DSN).
   
   For 32-bit Linux, it might look something like this:
   
   [MyUltraLightDSII]
   Description=Sample 32-bit SimbaEngine MyUltraLightDSII
   Driver=[INSTALLDIR]/SimbaEngineSDK/9.3/Examples/Source/MyUltraLight/Bin/Linux_x86/libMyUltraLight_debug.so
   Locale=en-US

   For 64-bit Linux, it might look something like this:
   
   [MyUltraLightDSII]
   Description=Sample 64-bit SimbaEngine MyUltraLightDSII
   Driver=[INSTALLDIR]/SimbaEngineSDK/9.3/Examples/Source/MyUltraLight/Bin/Linux_x8664/libMyUltraLight_debug.so
   Locale=en-US

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]
MyUltraLightDSIIDriver=Installed
```

6. Add a new section with a name that matches the new driver name. For example, it might look something like this:

For 32-bit Linux, it might look something like this:

```
[MyUltraLightDSIIDriver]
Driver=[INSTALLDIR]/SimbaEngineSDK/9.3/Examples/Source/MyUltraLight/Bin/Linux_x86/libMyUltraLight_debug.so
```

For 64-bit Linux, it might look something like this:

```
[MyUltraLightDSIIDriver]
Driver=[INSTALLDIR]/SimbaEngineSDK/9.3/Examples/Source/MyUltraLight/Bin/Linux_x8664/libMyUltraLight_debug.so
```

7. Copy the .simba.ultralight.ini file in your home directory so it has the name .simba.myultralight.ini

**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: `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 MyUltraLightDSII DSN.
4. To connect to your data source, type: `DSN=MyUltraLightDSII;UID=a;PWD=b`. A prompt that says “SQL>” appears.
5. Type a SQL command to query your database. For example, `SELECT *` from ULResultSet. This will output a simple result set.
6. To quit iodbctest, at the prompt, type `quit`.

You can also use a debugger, such as gdb, with the iodbctest utility. To use gdb test your driver and hit a breakpoint, do the following:

1. Type `gdb iodbctest` to start the debugger.
2. Type `break Simba::DSI::DSIDriverFactory` This will set a breakpoint at the DSIDriverFactory() function at line 31 in the Main_Unix.cpp file. This is a good breakpoint to start with because this function runs as soon as the Driver Manager loads the ODBC driver. If you would rather set a different
breakpoint, you can view the source code in this directory:


3. Type run DSN=MyUltraLightDSII;UID=a;PWD=b.

The program runs until it hits the breakpoint.

**Note:** When using the gdb debugger with an application, your ODBC driver is not loaded until the application is running and you make a connection. This means that you can set breakpoints before it is loaded or after, depending on which breakpoint you want to hit.

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

**Summary – Day One**

At this point, you have completed the following tasks:

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

**Day Two**

Today’s goal is to customize your driver, enable logging and establish a connection to your data store. To accomplish this you will visit TODO items 1 to 7.

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

**Construct a driver singleton**

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

The **DSIDriverFactory()** implementation in **Main_Windows.cpp** is the main hook that is called from Simba’s ODBC layer to create an instance of your DSI implementation. This method is called as soon as the Driver Manager calls **LoadLibrary()** on your ODBC driver.

1. Launch Microsoft Visual Studio.
2. Click **File > Open > Project/Solution**.
3. Navigate to
   [INSTALLDIR]\SimbaEngineSDK\9.3\Examples\Source\<YourProjectName>\Source
   and then open the <YourProjectName>_VS201x.vcproj file.

4. Rebuild your solution and the double click the TODO #1 message to jump to the relevant
   section of code.
   The Main_Windows.cpp file opens.

5. Look at the DSIDriverFactory() implementation.

6. Next, specify the location that will be used when reading driver settings from the registry.
   This change is related to rebranding. Replace DRIVER_WINDOWS_BRANDING with
   something like “Company\Driver” where “Company” is your company name and “Driver” is
   the name of your driver in the following line:

   SimbaSettingReader::SetConfigurationBranding(DRIVER_WINDOWS_BRANDING);

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

7. You may want to add processing at this point if you are building a commercial driver.

8. Click Save.

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

Set the driver properties

   TODO #2: Set the driver properties.

1. Double click the TODO #2 message to jump to the relevant section of code.
   The ULDriver.cpp file opens. Look at SetDriverPropertyValues() where you will set up
   the general properties for your driver.

2. Change the DSI_DRIVER_DRIVER_NAME setting. Set this to the name of your driver. (The
   same name you used to replace “UltraLightDSII” in Day One).

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

   - DSI_DRIVER_STRING_DATA_ENCODING – The encoding of char data within the data
     store. The default value is ENC_UTF8.

   - DSI_DRIVER_WIDE_STRING_DATA_ENCODING – The encoding of wide character data
     within the data store. The default is ENC_UTF16_LE.

Set the logging details

   TODO #3: Set the driver-wide logging details.
   TODO #4: Set the connection-wide logging details.

1. Double click the TODO #3 message to jump to the relevant section of code.
2. Change the driver log’s file name.

3. Double click the TODO #4 message to jump to the relevant section of code.

4. The connections currently use the same log file as the driver, you may choose to have each connection create a separate log file. If so, change the code to create a DSILog with a unique log file name.

5. Click Save All.

**Note:** By default, the SimbaEngine UltraLight Driver maintains one kind of log file for the entire driver. If you require more fine grained logging, then consider one for all driver-based calls and one for each connection created as noted in step 4, above.

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

### Check the connection settings

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

When the Simba ODBC layer is given a connection string from an ODBC-enabled application, the Simba ODBC layer parses the connection string into key-value pairs. Then, the entries in the connection string and the DSN are sent to the `ULConnection::UpdateConnectionSettings()` method which is responsible for verifying that all of the required, and any optional, connection settings are present.

For example, the connection string “DSN=UltraLight;UID=user;” will be broken down into key-value pairs and passed in via the DSIConnSettingRequestMap parameter. In this case that map would contain two entries: {DSN, UltraLight} and {UID, user}. If a DSN was specified, then the DSN value is removed from the map and any entries that are stored in the preconfigured DSN are inserted into the map. Once the map has been created with all the key-value pairs from the connection string and DSN, this map is passed down to the DSII.

1. Double click the TODO #5 message to jump to the relevant section of code.

2. The `UpdateConnectionSettings()` function should validate that the key-value pairs in `in_connectionSettings` are sufficient to create a connection, and any settings that are not present should be added to the `DSIConnSettingResponseMap` parameter.

   The `VerifyRequiredSetting()` or `VerifyOptionalSetting()` utility functions can be used to perform this verification and will add missing settings to `DSIConnSettingResponseMap` for you. If any of the values received are invalid, you should throw an `ErrorException` seeded with `DIAG_INVALID_AUTH_SPEC`.

   For example, the UltraLight driver verifies that the entries within `in_connectionSettings` are sufficient to create a connection, by using the following code:

   ```
   VerifyRequiredSetting(UL_UID_KEY, in_connectionSettings, out_connectionSettings);
   VerifyRequiredSetting(UL_PWD_KEY, in_connectionSettings, out_connectionSettings);
   VerifyOptionalSetting(UL_LNG_KEY, in_connectionSettings, out_connectionSettings);
   ```
The UltraLight driver requires a user ID and password, and can optionally take in a language (not currently used).

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

Customize the DriverPrompt Dialog

TODO #6: Customize DriverPrompt Dialog.

Depending on how the connection was initiated by the application, the SDK may call ULConnection::PromptDialog() to allow the user to specify more information. In general, if there are any required settings present in the DSIConnSettingResponseMap, then PromptDialog() will be called. Note that, if the application requests, PromptDialog() may not be called in this case or may be called even if there are no settings in the DSIConnSettingResponseMap.

ULConnection::PromptDialog() displays a configuration dialog box which is displayed by the Windows ODBC Data Source Administrator when configuring the driver.

The method takes in the following:

- **in_connResponseMap**: a connection response map which can be populated with settings which haven't been entered by the user. This is then used by the driver to notify the user that information is missing. Currently this variable is unused in the sample.
- **io_connectionSettings**: a connection settings map which is populated by the dialog with settings entered by the user.
- **in_parentWindow**: the handle to the parent Window to make the prompt window a child of.
- **in_promptType**: an enum specifying if only required fields are to be available, or if optional fields should be available as well. In the UltraLight driver, the language is an optional field.

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

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

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

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

**TODO #7: Establish A Connection.**

Once `ULConnection::UpdateConnectionSettings()` returns `out_connectionSettings` without any required settings (if there are only optional settings, a connection can still occur), the Simba ODBC layer will call `ULConnection::Connect()` passing in all the connection settings received from the application.

During `Connect()`, you should have all the settings necessary to make a connection as verified by `UpdateConnectionSettings()`. You can use the utility functions `GetRequiredSetting()` and `GetOptionalSetting()` to request the required and optional settings for your connection, and attempt to make an actual connection.

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

You have now authenticated the user against your data store.

Day Three

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

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

These catalog functions are represented in the DSI by metadata sources, one for each of the catalog functions.

Create and return metadata sources

**TODO #8: Create and return your Metadata Sources.**

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

2. Navigating the structure on a row-by-row basis: Move()

3. Retrieving data: GetMetadata() (See Data Retrieval, below for a brief overview of data retrieval). Each column in the metadata source will be represented by a DSIOutputMetadataColumnTag which is passed into GetMetadata().

Handle DSI_TYPE_INFO_METADATA

The underlying ODBC catalog function SQLGetTypeInfo is handled as follows:

1. When called with DSI_TYPE_INFO_METADATA, UDataEngine::MakeNewMetadataTable() will return an instance of UlTypeInfoMetadataSource().

2. The SimbaEngine UltraLight Driver example exposes support for all data types, but due to its underlying file format, it is constrained to support only the following types:

   SQL_BIGINT     SQL_BINARY     SQL_BIT
   SQL_CHAR       SQL_DECIMAL    SQL_FLOAT
   SQL_DOUBLE     SQL_INTEGER    SQL_LONGVARBINARY
   SQL_LONGVARCHAR SQL_LONGVARCHAR SQL_NUMERIC
   SQL_REAL       SQL_SMALLINT   SQL_TINYINT
   SQL_TYPE_DATE  SQL_TYPE_TIME SQL_TYPE_TIMESTAMP
   SQL_VARBINARY  SQL_VARCHAR   SQL_WCHAR
   SQL_WVARCHAR

3. For your driver, you may need to change the types returned and the parameters for the types in UlTypeInfoMetadataSource::InitializeData(). Populate the m DataTypes vector in this method, which defines the collection types that are supported along with their parameters.

Handle the other MetadataSources

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

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

• **ULTablesMetadataSource**: returns metadata about all of the tables in the data source. The sample hard codes and returns information for the hard coded person table to demonstrate how to return table metadata.

• **ULColumnsMetadataSource**: returns metadata for the columns in the data source. The sample hard codes and returns information for the three columns in the person table consisting of the name column, an integer column, and a numeric column.

2. When called with any other DSIMetadataTableId, which doesn’t correspond to these tables, `ULDataEngine::MakeNewMetadataTable()` returns a new instance of `DSIEEmptyMetadataSource` to indicate that no metadata is available for the specified table ID.

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

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

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

**Day Four**

Today’s goal is to enable data retrieval from within the driver. We will cover the process of preparing a query, providing parameter information, implementing a query executor, and implementing a result set.

**Prepare and execute a query**

TODO #9: Prepare a Query.

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

For demonstration purposes, the default implementation of `ULDataEngine::Prepare()` performs a very simple preparation by searching for the substrings “select” and “?” in the query. If “select” is found, then it is assumed that the caller wants to search for rows of data and a result set is therefore returned. If “select” is not found, then it is assumed that the caller wants to retrieve the number of rows and so a row count is therefore returned. If “?” is present, then the statement is assumed to be parameterized and therefore `ULDataEngine::PopulateParameters()` will populate parameters as described below. In your implementation you would replace this with more sophisticated logic or pass the query to the data source for preparation.

TODO #10: Implement an IQueryExecutor.
The ULQueryExecutor object returned by the ULDataEngine::Prepare() method is an implementation of IQueryExecutor which, as the name suggests, executes a query. After preparing a query, an application may execute it multiple times, in which case a single IQueryExecutor would be created by the prepare and would then be used for each execution.

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

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

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

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

TODO #11: Provide Parameter Information.

ULQueryExecutor::PopulateParameters() method is where parameter information is specified when the application calls SQLPrepare. The default implementation shows how to register input, input/output, and output-only parameters. Modify this method as required to register parameters appropriate for your queries.

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

TODO #12: Implement Query Execution

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

In your implementation, the Execute() method should begin by serializing parameters (stored in in_inputParamSetter) into a form that the data source can consume. Once this has been done then the data source should then be instructed to execute the statement, after which the results should be placed into the in_outputParamSetIter parameter.

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

TODO #13: Implement your DSISimpleResultSet
The final step in returning data is to implement a DSISimpleResultSet. The sample contains an implementation called ULResultSet which returns a hardcoded set of people’s names.

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

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

In the example, ULResultSet::MoveToNextRow() simply increments an row iterator so this should be replaced in your implementation with code that delegates this to the data source.

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

Day Five

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

Configure error messages

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

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

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

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

The vendor name is prepended to all error messages that are visible to applications. The default vendor name is Simba. To set the vendor name:

1. Double click the TODO #15 message to jump to the relevant section of code.
2. Set the vendor name as shown in the commented code.
Finishing touches

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

1. Rename all files and classes in the project to have the two-letter abbreviation you chose as part of TODO #14.

2. Create a driver configuration dialog. This dialog is presented to the user when they use the ODBC Data Source Administrator to create a new ODBC DSN or configure an existing one. The C++ SimbaEngine UltraLight Driver project contains an example ODBC configuration dialog that you can look at, as an example. You can find the source in the SimbaEngine UltraLight Driver Visual Studio project.

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

   IMPORTANT: If you are using 64-bit Windows with 32-bit applications, you must use the 32-bit ODBC Data Source Administrator. You cannot access the 32-bit ODBC Data Source Administrator from the start menu or control panel in 64-bit Windows. Only the 64-bit ODBC Data Source Administrator is accessible from the start menu or control panel. On 64-bit Windows, to launch the 32-bit ODBC Data Source Administrator you must run C:\WINDOWS\SysWOW64\odbcad32.exe. See Appendix A: ODBC Data Source Administrator on Windows 32-Bit vs. 64-Bit on page 30 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 the SimbaEngine SDK by modifying and customizing the UltraLight 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 64-Bit Windows

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

Data Source Names

To connect your driver to your database, the 32-bit ODBC Driver Manager on 64-bit Windows uses Data Source Name registry keys in HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBC.INI. Each key includes string values to define the name of the Driver, a Description to help you clearly identify each registry key, and a Locale to specify the language. The keys that are relevant to the C++ examples discussed in this document are:

- **UltraLightDSII** which must include the following string values:
  - Driver: UltraLightDSII_Driver
  - Description: Sample 32-bit SimbaEngine UltraLight DSII
  - Locale: en-US

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

- **ODBC Data Sources** which must include the following string values:
  - UltraLightDSII: UltraLightDSII_Driver

Driver Locations

To define each driver and its setup location, the 32-bit ODBC Driver Manager on 64-bit Windows uses registry keys created in HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBCINST.INI. Each key includes three string values to define the location of the Driver, its Setup location, and the Description to help you clearly identify each registry key. The three keys that are relevant to the C++ examples discussed in this document are:

- **UltraLightDSII_Driver** which includes the following key names and values:
  - Driver: [INSTALLDIR]\Examples\Builds\Bin\Win32\Release_MTDLL\UltraLightDSII_MTDLL.dll
  - Setup: [INSTALLDIR]\Examples\Builds\Bin\Win32\Release_MTDLL\UltraLightDSII_MTDLL.dll
o **Description:** Sample 32-bit SimbaEngine UltraLight DSII

There is another registry key at the same location called ODBC Drivers, indicating which drivers are installed. String values that correspond to each driver must also be added to it:

- **ODBC Drivers** which includes the following string values:
  - **UltraLightDSII.Driver:** Installed

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

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

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

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

### Data Source Names

To connect your driver to your database, the 64-bit ODBC Driver Manager on 64-bit Windows uses Data Source Name registry keys in `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI`. Each key includes three string values to define the name of the **Driver**, a **Description** to help you clearly identify each registry key, and a **Locale** to specify the language. The three keys that are relevant to the C++ examples discussed in this document are:

- **UltraLightDSII** which must include the following string values:
  - **Driver:** UltraLightDSII.Driver
  - **Description:** Sample 64-bit SimbaEngine UltraLight DSII
  - **Locale:** en-US

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

- **ODBC Data Sources** which must include the following string values:
  - **UltraLightDSII:** UltraLightDSII.Driver
Driver Locations

To define each driver and its setup location, the 64-bit ODBC Driver Manager on 64-bit Windows uses registry keys created in `HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI`. Each key includes three string values to define the location of the **Driver**, its **Setup** location, and the **Description** to help you clearly identify each registry key. The three keys that are relevant to the C++ examples discussed in this document are:

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

There is another registry key at the same location called **ODBC Drivers**, indicating which drivers are installed. String values that correspond to each driver must also be added to it:

- **ODBC Drivers** which includes the following string values:
  - **UltraLightDSII**Driver: **Installed**
Appendix C: Data Retrieval

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

1. Each MetadataSource implementation of GetMetadata()
2. DSISimpleResultSet::RetrieveData()

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

In addition, both methods accept the following three parameters:

1. in_data

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

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

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

2. in_offset

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

3. in_maxSize

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

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

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

Fixed length types

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

```
SQL_BIT
SQL_DATE
SQL_DECIMAL
SQL_DOUBLE
SQL_GUID
SQL_FLOAT
SQL_INTEGER
SQL_INTERVAL_DAY
SQL_INTERVAL_DAY_TO_HOUR
SQL_INTERVAL_DAY_TO_MINUTE
SQL_INTERVAL_DAY_TO_SECOND
SQL_INTERVAL_HOUR
SQL_INTERVAL_HOUR_TO_MINUTE
SQL_INTERVAL_HOUR_TO_SECOND
SQL_INTERVAL_MINUTE
SQL_INTERVAL_MINUTE_TO_SECOND
SQL_INTERVAL_MONTH
SQL_INTERVAL_SECOND
SQL_INTERVAL_YEAR
SQL_INTERVAL_YEAR_TO_MONTH
SQL_NUMERIC
SQL_REAL
SQL_SBIGINT
SQL_SINTEGER
SQL_SMALLINT
SQL_SSMALLINT
SQL_STINYINT
SQL_TINYINT
SQL_TIME
SQL_TIMESTAMP
SQL_TYPE_DATE
SQL_TYPE_TIME
SQL_TYPE_TIMESTAMP
SQL_UBIGINT
SQL_UINTeger
SQL_USMALLINT
SQL_UTINYINT
```
More information on Date, Time and DateTime types

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

Simple Fixed-Length Data Example

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

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

Variable Length Types

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

- SQL_BINARY
- SQL_CHAR
- SQL_LONGVARBINARY
- SQL_LONGVARCHAR
- SQL_VARBINARY
- SQL_VARCHAR
- SQL_WCHAR
- SQL_WLONGVARCHAR
- SQL_WVARCHAR

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

Simple Variable-Length Data Example

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

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

NULL Values

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

in_data->SetNull(true);
Appendix D:  C++ Server Configuration

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

IMPORTANT: The information in this section only applies if you are using 32-Bit Windows. If you are using 64-bit Windows (with either 32-bit or 64-bit applications), the file paths must be configured appropriately. Please see Appendix B: Windows Registry 32-Bit vs. 64-Bit on page 31 for details.

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

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

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

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

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

1. Build UltraLight using the debug (or release) server configuration:
   
   BUILDSERVER=exe make -f UltraLight.mak debug

2. Configure the server as required in the other sections of the .simbaserver.ultralight.ini file.

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

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2010 International Business Machines Corporation and others

All rights reserved.

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

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

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

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

OpenSSL

Copyright (c) 1998-2008 The OpenSSL Project. All rights reserved.

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

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

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

3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
   "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)

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

5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.

6. Redistributions of any form whatsoever must retain the following acknowledgment:
   "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)

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

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

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

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

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

stringencoders

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

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.