Simba Engine SDK 9.1

Build an ADO.NET Provider in 5 Days

Last Revised: May 2013

Simba Technologies Inc.
Table of Contents

Build an ADO.NET provider in five days ................................................................. 1
  About the SimbaEngine SDK ............................................................................... 1
  About the DotNetUltraLight sample solutions .................................................. 2
  About the DotNetUltraLight provider ............................................................... 2

Day One .................................................................................................................... 4
  Install the SimbaEngine SDK ............................................................................ 4
  Build the DotNetUltraLight example provider .................................................. 4
  Install the provider's assembly into the Global Assembly Cache ..................... 4
  Install the other required assemblies into the GAC ......................................... 5
  Configure the .NET Framework to locate the provider ..................................... 6
  Test the data source ............................................................................................ 7
  Set up a new project to build your own ADO.NET provider .............................. 7
  Build your new provider ..................................................................................... 8
  Update the Global Assembly Cache .................................................................. 8
  Update the machine.config file .......................................................................... 9
  Test your new data source ................................................................................... 9

Day Two ................................................................................................................... 10
  View the list of TODO messages ....................................................................... 10
  Rename the Simba.ADO.Net sub-classes ............................................................ 11
  Construct the IDriver instance ......................................................................... 11
  Set the properties ............................................................................................... 11
  Create properties for the connection string keys ............................................. 12
  Check the connection settings .......................................................................... 12
  Establish a connection ....................................................................................... 12

Day Three ............................................................................................................... 13
  Create and return metadata sources ................................................................. 13

Day Four .................................................................................................................. 15
  Prepare a query .................................................................................................... 15
  Execute a prepared query .................................................................................. 15
  Provide parameter information ......................................................................... 15
  Implement query execution ............................................................................... 16
  Retrieve the query results .................................................................................. 16

Day Five .................................................................................................................. 18
  Set the vendor name ......................................................................................... 18
  Set the branding ................................................................................................. 18

Appendix A: Data Retrieval .................................................................................... 19

Third Party Trademarks ......................................................................................... 20
Build an ADO.NET provider in five days

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

About the SimbaEngine SDK

The SimbaEngine SDK contains a complete implementation of the ADO.NET specification, which provides a standard interface to which any ADO.NET 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. This will be used by common reporting applications to access your data store when SimbaEngine executes an SQL statement. The diagram below shows how your custom-designed DSI implementation (DSII) connects directly to your data source.

Figure 1: High-level view of SimbaEngine

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. Full documentation for the SimbaEngine SDK is available on the Simba website at http://www.simba.com/odbc-sdk-documents.htm.
About the DotNetUltraLight sample solutions

The DotNetUltraLight sample contains two solutions that each use different APIs:

- The "DotNetUltraLight_Provider" solution uses the Simba.ADO.NET API. The DotNetUltraLight_Provider VS2012.sln file implements a provider that is written entirely in C#, providing an ADO.NET interface. It is a sample DSI implementation of an ADO.NET provider, which accesses a sample in-memory data source. The Simba SQLEngine is not used with the ADO.NET provider.
  **Note:** This is the solution that is described in this document.

- The "DotNetUltraLight_Driver" solution uses Simba’s C++ to C# bridge (CLIDSI) API. The DotNetUltraLight_Driver VS2012.sln file implements a driver using a mixture of C# and C++, providing an ODBC interface or SimbaServer executable for use with any of the SimbaClient drivers.

About the DotNetUltraLight provider

The DotNetUltraLight sample provider helps you to prototype a DSI implementation for your own data store so you can learn how the SimbaEngine SDK works. You can also use it as the foundation for a 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.

In the DotNetUltraLight sample provider, there is a pattern of class relationships, headed by IResultSet and anchored by your MetadataSource classes (For example, ULTablesMetadataSource) and Table classes (For example, ULPersonTable).

For data retrieval, your Reader class interacts directly with your data store to retrieve the data and deliver it to the Table class on demand. The Reader class should take care of caching, buffering, paging, and all the other techniques that speed data access. Implementing metadata access is a bit more complicated. There are several Metadata Sources that you can implement, but as a starting point, to make your provider work properly, you only need to implement the following Metadata Sources:

- Catalog only
- Schema only
- Columns
- Tables
- Type Information
A typical design pattern for a DSI implementation is shown in the following UML diagram:

Figure 2: Design pattern for a DSI implementation.
Day One

Today's task is to set up the development environment and project files for your provider. By the end of the day, you will have compiled and tested your ADO.NET provider.

Install the SimbaEngine SDK

**Note:** If you have a previous version of the SimbaEngine SDK installed, you must uninstall it before you install 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 DotNetUltraLight example provider

**Note:** Visual Studio 2012 is used for the examples, but newer versions are also supported.

1. Launch Microsoft Visual Studio.
2. Click **File > Open > Project/Solution**.
3. Navigate to \[INSTALL_DIRECTORY\]\SimbaEngineSDK\9.1\Examples\DotNetUltraLight\Source and then open the DotNetUltraLight_Provider_VS2012.sln file.  
The default \[INSTALL_DIRECTORY\] is C:\Simba Technologies.
4. Click **Build > Configuration Manager** and make sure that the active solution configuration is set to “Debug” and then click **Close**.
5. Click **Build > Build Solution** or press F7 to build the provider.

Install the provider's assembly into the Global Assembly Cache

Each time you build the DLL, it must be installed into the Global Assembly Cache (GAC) before it can be used. To run the Global Assembly Cache tool, use the Visual Studio Command Prompt. You must run this command as an administrator.

1. On the taskbar, click **Start > All Programs > Microsoft Visual Studio > Visual Studio Tools**.
2. Right-click **Visual Studio Command Prompt** and select "Run as administrator".
3. Change to the directory that contains the DLL file that you just built. For example, type a command that is similar the following:
   
   ```
   cd \[INSTALL_DIRECTORY\]\SimbaEngineSDK\9.1\Examples\DotNetUltraLight\Bin\win\debug
   ```
4. Type the following command to install the assembly into the GAC:
   
   ```
   gacutil.exe /i Simba.UltraLight.Provider.dll
   ```
You will see the message, "Assembly successfully added to the cache" if the operation was successful.

**Note:** If that assembly was already installed in the GAC, you must uninstall it before you try to install it again. To uninstall the assembly from the GAC before installing it again, run the following command (as administrator): `gacutil.exe /u Simba.UltraLight.Provider`

### Install the other required assemblies into the GAC

In addition to the DLL of your provider, `Simba.ADO.Net.dll` and `Simba.DotNetDSI.dll` must be installed in the GAC. These files were installed in the GAC during SDK installation.

#### Simba.ADO.Net assembly

In order to check the GAC for the `Simba.ADO.Net` assembly, run the following command:

```cmd
gacutil.exe /l Simba.ADO.Net
```

If the assembly is already installed in the GAC, then you will see the message "Number of items = 1" and you can move on to checking the next DLL. However, if the assembly is not installed in the GAC, then you will see the message "Number of items = 0" and you must install the assembly manually. To do this, run the following command:

```cmd

```

#### Simba.DotNetDSI assembly

In order to check the GAC for the `Simba.DotNetDSI` assembly, run the following command:

```cmd
gacutil.exe /l Simba.DotNetDSI
```

If the assembly is already installed in the GAC, then you will see the message "Number of items = 1" and you can move on to checking the other DLL. However, if the assembly is not installed in the GAC, then you will see the message "Number of items = 0" and you must install the assembly manually. To do this, run the following command:

```cmd

```

#### Simba.ADO.Net.DDEX assembly

The Data Designer Extensibility (DDEX) assembly is used to hook into Analysis Services and Visual Studio. It maps from the Microsoft models to the provider models that are supplied by the SDK.
In order to check the GAC for the Simba.ADO.Net.DDEX assembly, run the following command:

gacutil.exe /l Simba.ADO.Net.DDEX

If the assembly is already installed in the GAC, then you will see the message "Number of items = 1" However, if the assembly is not installed in the GAC, then you will see the message "Number of items = 0" and you must install the assembly manually. To install this assembly, type the following command:

gacutil.exe /i "[INSTALL_DIRECTORY]\SimbaEngineSDK\9.1\DataAccessComponents\Bin\win\release\Simba.ADO.Net.DDEX.dll"

Configure the .NET Framework to locate the provider

1. Open a text editor as an administrator. For example, to open WordPad as an administrator, click Start > All Programs > Accessories and then right-click WordPad and click Run as administrator.

2. In the text editor, open the machine.config files for the version of the Microsoft .NET framework that you are using.

For example, if you were using Microsoft Visual Studio 2010, you would modify these files:

C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\machine.config
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\machine.config

If you were using Microsoft Visual Studio 2012, you would modify these files:

C:\Windows\Microsoft.NET\Framework\v2.0.50727\Config\machine.config
C:\Windows\Microsoft.NET\Framework64\v2.0.50727\Config\machine.config

3. Locate the <system.data><DbProviderFactories> node.

4. Insert the following node within the <DbProviderFactories> node:


Note: Do not delete the other "name" nodes that may already be present in the file.

5. At the command prompt, run the following command:

   gacutil.exe /l Simba.UltraLight.Provider

You will see a message similar to this:

The Global Assembly Cache contains the following assemblies:
   Simba.UltraLight.Provider, Version=9.1.0.1000, Culture=neutral, PublicKeyToken=85df83a0046b8966, processorArchitecture=MSIL
This shows you the "invariant" name of the provider, which is "Simba.UltraLight.Provider", the Version and the PublicKeyToken.

6. In both machine.config files, for the new node that you just added, adjust the Version and PublicKeyToken to match the information from the gacutil message in the previous step. The XML node will look something like this:

```xml
<system.data>
  <DbProviderFactories>
    ...
    <add name="UltraLightDSII Data Provider" invariant="Simba.UltraLight.Provider" description=".NET Framework Data Provider for UltraLightDSII"
      type="Simba.UltraLight.ULDotNetFactory, Simba.UltraLight.Provider, Version=9.1.0.1000, Culture=neutral, PublicKeyToken=85df83a0046b8966"/>
    ...
  </DbProviderFactories>
</system.data>
```

7. Save and close both files.

Test the data source

To test the provider, you can use the Simba ADO.NET Provider Test Program that is provided with the DotNetUltraLight example.

1. Open a Windows command prompt.
2. Type the following command to launch the Simba ADO.NET Provider Test Program:

   ```cmd
   "[INSTALL_DIRECTORY]\SimbaEngineSDK\9.1\Examples\DotNetUltraLight\Bin\win\debug\TestApp.exe" Simba.UltraLight.Provider UID=na; PWD=na
   ```

   The UID and PWD command line options must be specified but, because the provider does not require a user id and password, you can just type any value for them.

   The test program connects to the Simba.UltraLight.Provider.

3. Enter the following test query:

   ```sql
   SELECT * FROM person
   ```

   The schema data and the results of the SQL query are displayed.

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

Set up a new project to build your own ADO.NET provider

Now that you have built the example provider, you are ready to set up a development project to build your own ADO.NET provider.
**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 provider. If you have modified the sample provider, this will not be possible.

1. In your Windows Explorer window, copy the 
   \[INSTALL_DIRECTORY\]SimbaEngineSDK\9.1\Examples\DotNetUltraLight directory 
   and paste it to the same location. This will create a new directory called 
   "DotNetUltraLight - 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 the Source directory of your new copy and then right-click the 


4. In the Microsoft Visual Studio menu, click View > Solution Explorer.

5. Using the Solution Explorer, rename the DotNetUltraLight_Provider_VS2012 solution to 
   \(<YourProjectName>_Provider_VS2012\).

6. Rename the C# project UltraLight_Provider_VS2012 to 
   \(<YourProjectName>_Provider_VS2012\).

7. In the Microsoft Visual Studio menu, click Project and then click 
   \(<YourProjectName>_Provider_VS2012 Properties\).

8. In the Assembly name text box, replace Simba.UltraLight.Provider with 
   \(<YourCompanyName>\.\<YourProjectName>\.Provider\).

9. Click File > Save All.

**Build your new provider**

Click Build > Build Solution or press F7 to build the provider.

**Update the Global Assembly Cache**

Each time you build the DLL, it must be installed to the Global Assembly Cache (GAC).


2. Right-click Visual Studio Command Prompt and select "Run as administrator".

3. Change to the directory that contains the DLL file that you just built. For example, type a 
   command that is similar the following:

   ```
   cd \[INSTALL_DIRECTORY\]SimbaEngineSDK\9.1\Examples\<YourProjectName>\Bin\win\debug
   ```
4. Type the following command to install the assembly into the GAC:
   
gacutil.exe /i <YourAssemblyName>.dll

   You will see the message, "Assembly successfully added to the cache" if the operation was successful.

   **Note**: Each time you make changes to your provider in the upcoming days you will need to uninstall and re-install your provider from the GAC. To uninstall your assembly from the GAC before installing it again, run the following command (as administrator):

   
gacutil.exe /u <YourAssemblyName>

**Update the machine.config file**

- Add a new node to the `machine.config` file.
- You will also want to change the name, invariant name, description, as well as the assembly name in the type field of the `<add>` node you are adding to `machine.config`.
- Take note of the invariant name you set as this is how you will tell the test app to use your provider.
- Each time you make changes to your provider in the upcoming days you will need to uninstall and re-install your provider from the GAC but will not need to change `machine.config` unless instructed to.

   For detailed instructions, refer to the earlier section, "Configure the .NET Framework to locate the provider".

**Test your new data source**

To test your new provider, use the Simba ADO.NET Provider Test Program again.

1. Open a Windows command prompt.
2. Type the following command to launch the Simba ADO.NET Provider Test Program:
   
   "[INSTALL_DIRECTORY]\SimbaEngineSDK\9.1\Examples\DotNetUltraLight\Bin\win\debug\TestApp.exe" <YourAssemblyInvariantName> UID=na; PWD=na

   The UID and PWD command line options must be specified but, because the provider does not require a user id and password, you can just type any value for them.

   The test program connects to your provider.

3. Enter the following test query:
   
   SELECT * FROM person

   The schema data and the results of the SQL query are displayed.

   If there were no problems, you are now ready to customize your provider.
Day Two

Today’s goal is to customize your provider, enable logging and establish a connection to your data store. In the DotNetUltraLight provider, the areas of the code that you need to change are marked with “TODO(ADO)” messages along with a short explanatory message.

Note: These “TODO(ADO)” messages are distinct from “TODO(ODBC)” messages that are for a different solution. For the purposes of this guide, you can disregard the “TODO(ODBC)” messages.

Most of the areas of the code that you need to modify are for productization. These are things like naming the provider, setting the properties that configure the provider, and naming the 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 SimbaEngine SDK. Because the DotNetUltraLight provider 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 provider work against your own data store.

View the list of TODO messages

1. Go to Microsoft Visual Studio.
2. Click Edit > Find and Replace > Find in Files.
3. In the Find and Replace window, in the Find what text box, type TODO(ADO).
4. Click Find All.

The results are displayed in the Find Results output window. The list of TODO messages is as follows:

<table>
<thead>
<tr>
<th>TODO(ADO)</th>
<th>#1: Rename the Simba.ADO.Net sub-classes.</th>
<th>(ULDotNetFactory.cs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TODO(ADO)</td>
<td>#2: Construct the IDriver instance.</td>
<td>(ULDotNetConnection.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#3: Set the driver properties.</td>
<td>(UDriver.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#4: Create properties for the connection string keys.</td>
<td>(ULDotNetConnectionStringBuilder.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#5: Check connection settings.</td>
<td>(ULConnection.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#6: Establish a connection.</td>
<td>(ULConnection.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#7: Create and return your Metadata Sources.</td>
<td>(ULDataEngine.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#8: Prepare a query.</td>
<td>(ULDataEngine.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#9: Implement a QueryExecutor.</td>
<td>(ULQueryExecutor.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#10: Provide parameter information.</td>
<td>(ULQueryExecutor.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#11: Implement Query Execution.</td>
<td>(ULQueryExecutor.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#12: Implement your DSISimpleResultSet.</td>
<td>(ULPersonTable.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#13: Set the vendor name, which will be prepended to error messages.</td>
<td>(UDriver.cs)</td>
</tr>
<tr>
<td>TODO(ADO)</td>
<td>#14: Set the branding of the registry key to read configuration from.</td>
<td>(ULDotNetConnection.cs)</td>
</tr>
</tbody>
</table>
Over the next four days, you will be visiting each “TODO” and modifying the source code.

Today’s goal is to customize your provider including application and user-facing components that identify your provider, enable logging, and establish a connection to your data store. To accomplish this you will visit TODO items 1 to 6.

Rename the Simba.ADO.Net sub-classes

TODO(ADO)  #1: Rename the Simba.ADO.Net sub-classes.  (ULDotNetFactory.cs)

1. In Microsoft Visual Studio, open the file that contains the TODO #1 message.

2. Each of the classes in the DotNet folder of the solution explorer should be renamed for your provider. These classes are:
   - ULDotNetFactory
   - ULDotNetCommand
   - ULDotNetCommandBuilder
   - ULDotNetConnection
   - ULDotNetConnectionStringBuilder
   - ULDotNetDataAdapter
   - ULDotNetParameter

3. To rename them, use Visual Studio’s rename refactoring utility as follows. Select the class name then from the right-click menu, select Refactor -> Rename (Ctrl + R, Ctrl + R). For each class, choose a name replacing the prefix ULDotNet with your own prefix. You should also rename their filenames to correspond to the class name.

4. Click Save.

Construct the IDriver instance

TODO(ADO)  #2: Construct the IDriver instance.  (ULDotNetConnection.cs)

The CreateDSIDriverInstance method is the main entry point for Simba.ADO.Net to initialize your provider. This method is called once as soon as an application first tries to connect to your provider. There is nothing to change here right now, although you may want to add processing at this point for a commercial provider.

Set the properties

TODO(ADO)  #3: Set the driver properties.  (ULDriver.cs)

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

2. Change the DSI_DRIVER_DRIVER_NAME setting. Set this to the name of your provider.

   Note: You may want to revisit this section when fully productizing your provider.
Create properties for the connection string keys

**TODO(ADO) #4: Create properties for the connection string keys.**

Create properties for the connection string keys. Here you will rename or replace the existing properties UserName, Password, and Language with your own properties that may be used for connections. Take note of how each of the existing properties are implemented by storing and retrieving the value from the base class map accessor: `this[KeyString]`.

Check the connection settings

**TODO(ADO) #5: Check connection settings.**

Check the connection settings. Given a connection string from the ADO.NET application, the Simba.ADO.Net layer will parse the connection string into key-value pairs before calling ULConnection's `UpdateConnectionSettings()` method to validate its contents. This method should validate that the entries within the `requestSettings` are sufficient to create a connection. If not, you can ask for additional information from the application by specifying the additional settings in the return value.

Should any of the values received be invalid, you should throw an exception. Note however that you should only be checking that the values be in the correct form or within certain allowable ranges. Do not attempt to communicate with the data store yet to validate keys such as username and password. For your convenience, you can also use the utility functions supplied: `VerifyRequiredSetting()` and `VerifyOptionalSetting()`. If there are no further entries required, simply leave the returned dictionary empty.

Establish a connection

**TODO(ADO) #6: Establish a connection.**

Establish a connection. Once ULConnection's `UpdateConnectionSettings()` returns a dictionary without any required settings (if there are only optional settings, a connection can still occur), the Simba.ADO.Net layer will call ULConnection's `Connect()` passing in all the connection settings received from the application. This is where you should authenticate the user against your data store using the information provided within the `connectionSettings` parameter.

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

You have now authenticated the user against your data store.
Day Three

Today’s goal is to return the data used to return schema information to the ADO.NET application. The majority of all ADO.NET applications require the following schema names:

- DataTypes
- Tables
- Columns

Create and return metadata sources

ULDataEngine's MakeNewMetadataSource() is responsible for creating the sources to be used to return data to the ADO.NET application for the various schemas. Schemas are mapped to a unique MetadataSourceId, which is then mapped to an underlying IMetadataSource that you will implement and return. Each IMetadataSource instance is responsible for the following:

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

Handle MetadataSourceID.TypeInfo

The DataTypes schema is handled as follows:

1. When called with TypeInfo, ULDataEngine’s MakeNewMetadataSource() will return an instance of ULTypeInfoMetadataSource.

2. The example provider exposes support for all data types although its one table only contains the following types:
   - SQL_WVARCHAR
   - SQL_INTEGER
   - SQL_NUMERIC

3. For your provider, you may need to change the types returned and the parameters for the types in ULTypeInfoMetadataSource's InitializeDataTypes().
Handle the other MetadataSources

The other schemas are handled in a similar fashion to Type Information Metadata.

1. When called with any other MetadataSourceID, MakeNewMetadataSchema() will return the appropriate instance of an implementation of IMetadataSource as illustrated by the DotNetUltraLight provider.

2. Your implementations of IMetadataSource should query your data store to obtain the appropriate metadata and provide the means to iterate through that metadata and to return the metadata.

You can now retrieve type metadata from within your data store. You should be able to connect to your provider with the ADO.NET sample application and see the correct metadata returned. Using the sample application, when prompted for a query, instead enter one of the following schema names: “DataTypes”, “Tables”, or “Columns”.
Day Four

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

We will cover the process of preparing a query, executing the prepared query, retrieving the query result, retrieving the column information for the query result, and finally retrieving data.

Prepare a query

TODO (ADO) #8: Prepare a query. (ULDataEngine.cs)

ULDataEngine’s Prepare() is the entry point where the SimbaEngine SDK requests queries to be prepared. You must modify this method to perform the following:

- Send a request to your data store to prepare the query.
- Handle the response from your data store.
- Create an instance of your IQueryExecutor implementation containing whatever information is necessary to execute the query.

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

Execute a prepared query

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

TODO (ADO) #9: Implement a QueryExecutor. (ULQueryExecutor.cs)

You will need to modify the constructor of ULQueryExecutor to receive information from query preparation to be used for query execution. In the constructor, you must also update the Results property to be a list of IResult of the correct IResultSet or IRowCountResult types. These results should not contain actual data yet but may be used to retrieve column metadata before the query is executed.

Provide parameter information

TODO (ADO) #10: Provide parameter information. (ULQueryExecutor.cs)

If your data store is capable of handling query parameters, you will need to fill the ParameterMetadata list with relevant parameter metadata for the query. If the query contains no parameters, an empty list should be created.
Implement query execution

TODO(ADO)  #11: Implement Query Execution. (ULQueryExecutor.cs)

ULQueryExecutor’s Execute() is the entry point where the SimbaEngine SDK requests queries to be executed. You must modify this method to perform the following:

- Serialize all input parameters (if any) in a form that can be consumed by the data store.
- Send a request to your data store to execute the query.
- Retrieve all output parameters (if any) from the data store.
- Prepare to retrieve query results from the data store.

Retrieve the 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(ADO)  #12: Implement your DSISimpleResultSet. (ULPersonTable.cs)

ULPersonTable implements a simple in-memory table. 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.

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

- Return the columns defined for your table.
  - InitializeColumns(): This method must be modified so that, for each column defined in the query, you define the ColumnMetadata in terms of SQL types.

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

Get all the column information from your data store for the table
For Each Defined Column
{
    // Set the argument of the following method call to the SQL Type that
    // maps to the data store type of the column.
    TypeMetadata typeMetadata =
        TypeMetadata.CreateTypeMetadata(SqlType.VarChar);
    // Depending on SQL type, set different properties:
    if (character type)
    {
        typeMetadata.IntervalPrecision = m_settings.m_maxColumnSize;
    }
    else if (exact numeric type)
    {
        typeMetadata.Scale = scale;
    }
}
} // Create the column metadata.
DSIColumn columnMetadata = new DSIColumn(typeMetadata);
columnMetadata.Catalog = m_catalogName;
columnMetadata.Schema = m_schemaName;
columnMetadata.TableName = m_tableName;
columnMetadata.Name = "column name";
columnMetadata.Label = "localized column name";
columnMetadata.IsNull = Nullability.Nullable;
if (character type)
{
    columnMetadata.Size = m_settings.m_maxColumnSize;
}

// Add the column metadata to the list of column metadata.
m_columns.add(columnMetadata);
}

• Data Retrieval
  o MoveToNextRow()
  o GetData()

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

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

In the DotNetUltraLight Provider, ULPersonTable stores its data in an in-memory list of a
class specific to describing rows of this table. Each member variable in the RowData object
represents a column of data. The GetData method takes a column index and uses it to
determine from which member variable of the current row/object to retrieve data. See Data
Retrieval, for a brief overview of data retrieval.

  o DoCloseCursor()

This is a callback method called by the SimbaEngine SDK 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.

You can now execute queries and retrieve data from your data store. You should be able to use
the sample ADO.NET application to execute queries and to see the results returned from your
data store.
Day Five

Today’s goal is to start productizing your provider.

Set the vendor name

TODO(ADO)  #13: Set the vendor name, which will be prepended to error messages. (ULDriver.cs)

Most error messages generated within the Simba.ADO.Net and Simba.DotNetDSI components will have a vendor name or brand prepended to help identify the source of the error. Here you should uncomment the VendorName property and change the string it returns to be one to identify your brand or provider.

Set the branding

TODO(ADO)  #14: Set the branding of the registry key to read configuration from. (ULDotNetConnection.cs)

Change the string here to return the key name indicating where to read configuration values from in the registry. The default value of @"Simba\DotNetUltraLight" causes the config values to be read from “Software\Simba\DotNetUltraLight\Driver”. (Note that 32-bit applications on 64-bit platforms will read from “Software\Wow6432Node\Simba\DotNetUltraLight\Driver”.)

Finally, rename any remaining namespaces, files, and classes that contain the name “UltraLight” or abbreviation “UL”.

You are now done with all of the TODO’s in the project. You have created your own, custom ADO.NET provider using the SimbaEngine SDK by modifying and customizing the DotNetUltraLight sample provider.
Appendix A: Data Retrieval

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

1. Each IMetadataSource implementation of GetMetadata()
2. 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 SimbaEngine SDK will pass in the column index starting at 0.

In addition, both methods accept the following three parameters:

1. out_data

   The Object into which you must set your cell's value. The data you set must be represented as the Object or primitive data type that corresponds to the data type you set in the column metadata. For example, if a column is a SqlType.Integer, you must use a System.Int32. For a full list of the types used, see the documentation for Simba.DotNetDSI.DataEngine.SqlType. If your data is not stored as the appropriate type, you will need to write code to convert from your native format.

2. offset

   Some data types can be retrieved in parts. This value specifies where in the current column the value should be copied from. The value is usually 0.

3. maxSize

   The maximum size (in bytes) that can be copied into the type. For character or binary data, copying data over this amount can result in a data truncation warning, or worse, a heap-violation.
Third Party Trademarks

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

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

All rights reserved.

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

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

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

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

OpenSSL

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

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

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

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

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

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

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

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

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

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

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

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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND 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."