



class should take care of caching, buffering, paging, and all the other techniques that speed data access. Implementing metadata access is a bit more complicated, but it is not as bad as it looks. There are 13 sub-classes of `MetadataSource` that you can implement, but if you implement only the five most important and throw a `Not Supported` exception for the rest, your driver will work properly with Microsoft Excel!

As a last resort, if you are truly stuck, call us. Our support line is open from 8:00 AM to 6:00 PM Pacific Time, Monday to Friday. Dial 604-633-0008 and select 2. Or you can send us an e-mail at support@simba.com. We would be happy to answer your questions and get you going again.

Before you start you should be familiar with Microsoft's Visual Studio development environment, the C++ development language, and object orientated principals in general. The ODBC API Reference is available at [http://msdn.microsoft.com/en-us/library/ms714562\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms714562(VS.85).aspx).

Day One

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

Initial Set Up

Start by installing the SDK and running the example drivers:

1. Install SimbaEngine SDK to the default location: "`Program Files\Simba`" on Windows. The example drivers will be installed in `\My Documents\SimbaEngine 8.0 Projects\Examples`.
2. Take a few minutes to read the Getting Started Guide. It contains detailed information on the content of the SDK and introduces you to the architecture of the SimbaEngine SDK solutions.
3. The installer will create two DSNs in the registry for each of the example ODBC drivers – one for the release version of the ODBC driver and one for the debug version. There are two example Simba SQL Engine ODBC drivers: SimbaEngine Codebase Driver and SimbaEngine Quickstart Driver.

4. You are now ready to build, test and verify that the example drivers are working. You can use any ODBC application for this, such as Microsoft Access, Microsoft Excel or `ODBCTest32.exe (Unicode)`.

Build and Test the Example Drivers

1. Before you can compile and debug the examples, you must edit and merge the two registry (.reg) files located in your development folder. These files will update your registry with the required ODBC driver and DSN settings for the development version of the Quickstart driver.
2. The source for the example ODBC drivers is located in the Examples folder `\My Documents\SimbaEngine 8.0 Projects\Examples`.
3. Open the Visual Studio .NET 2005 solution of the example that you wish to build and test. Start with the SimbaEngine Quickstart Driver example.
4. From the Main Menu select `Build->Build Solution (F7)` to build the driver. By default, this will build the debug version of the driver.
5. Open any ODBC-enabled application (e.g.: ODBC Test) and attach Visual Studio to its process. Do this by selecting `Debug->Attach to Process...` from the Main Menu and clicking on the ODBC application you are using.
6. You should now be able to place breakpoints anywhere in the SimbaEngine Quickstart DSI implementation. A good breakpoint to start with is at `Main_Windows.cpp DSIDriverFactory()`. This code runs as soon as the Driver Manager loads the ODBC driver.
7. Execute an ODBC operation within the application. You should hit the breakpoint you created in the previous step and the Windows focus should switch to Visual Studio.
8. Perform the same steps with the SimbaEngine Codebase Example Driver.

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

Setting up the project

1. Copy the SimbaEngine Quickstart Driver directory and paste it to the same location. This will create a new directory called "Copy of SimbaEngineQuickstart".



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.

2. Open your new directory and rename the Quickstart_net2005.vcproj file located in the Source directory. This is the project file for your new ODBC driver, so name it accordingly.
3. Using a text editor, open the project file and replace every instance of "QuickstartDSII" in the source code with the name of your new ODBC driver. Then save and close the file.
4. Build the project to make sure everything compiles. At this point, the new DSII project is identical to the SimbaEngine Quickstart Driver example.
5. When you build your new project, you should see the following TODO messages appear in the Output window. You can open the Output window by selecting Debug->Windows->Output from the Main Menu:

TODO #1: Construct driver singleton.

TODO #2: Set the driver properties.

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

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

TODO #5: Check Connection Settings.

TODO #6: Establish A Connection.

TODO #7: Create and return your Metadata Sources.

TODO #8: Open A Table.

TODO #9: Update Messages xml file name.

TODO #10: Update Driver name.

TODO #11: Update Driver name.

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

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

Day Two

Today's goal is to customize your driver, enable logging and establish a connection to your data store. To accomplish this, you will visit TODO items 1 to 6.

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. For the purposes of prototyping, this TODO is purely informational. There is nothing to change here right now, although you may want to add processing at this point for a commercial driver.

TODO #2: Set the driver properties.

In QSDriver::QSDriver() you will set up general properties for your driver. At the very least, you will need to change:

- DSI_DRIVER_NAME – set this to the name of your driver.

Depending on the character sets or Unicode encoding used on your data store, you may also want to change:

- DSI_DRIVER_STRING_DATA_ENCODING – The encoding of char data within the data store. The default value is usually ENC_CP1252.
- DSI_DRIVER_WIDE_STRING_DATA_ENCODING – The encoding of wide character data within the data store. The default is UTF-16LE.

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

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

By default, the SimbaEngine Quickstart Driver maintains two kinds of log files: one for all driver-based calls and one for each connection created. Update these TODO's if you do not require such fine granularity in logging.



TODO #5: Check Connection Settings.

Given a connection string from the ODBC-enabled application, the Simba ODBC layer will parse the connection string into key/value pairs before calling `QSCONNECTION::UpdateConnectionSettings()` to validate its contents. This method should validate that the entries within `in_connectionSettings` are sufficient to create a connection. If not, you can ask for additional information from the ODBC-enabled application by adding the additional settings to the `out_connectionSettings`.

Should any of the values received be invalid, you should throw an `ErrorException` seeded with `DIAG_INVALID_AUTH_SPEC`. If there are no further entries required, simply leave `out_connectionSettings` empty.

TODO #6: Establish A Connection.

Once `QSCONNECTION::UpdateConnectionSettings()` returns an empty `out_connectionSettings`, the Simba ODBC layer will call `QSCONNECTION::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 `in_connectionSettings` parameter.

Should authentication fail, you should throw an `ErrorException` seeded with `DIAG_INVALID_AUTH_SPEC`.

Congratulations! You have now successfully authenticated the user against your data store.

Day Three

Today's goal is to return the data used to return catalog information to the ODBC-enabled application. 99.9% of all ODBC-enabled applications require the following ODBC catalog functions:

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

TODO #7: Create and return your Metadata Sources.

`QSDATAENGINE::MakeNewMetadataTable()` is responsible for creating the sources to be used to return data to the ODBC-enabled application for the various ODBC catalog functions. Each ODBC catalog function is mapped to a unique `DSIMETADATABLEID`, 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: *GetData()* (See the section below titled "Technical Note: Data Retrieval" for a brief overview of data retrieval).

Handling DSI_TYPE_INFO_METADATA

1. When called with `DSI_TYPE_INFO_METADATA`, `QSDATAENGINE::MakeNewMetadataTable()` will return an instance of `QSTYPEINFOMETADATASOURCE()`.
2. The SimbaEngine Quickstart Driver example exposes support for all data types, but due to its underlying file format it is constrained to support only the following types:

<code>SQL_VARCHAR</code>	<code>SQL_SMALLINT</code>	<code>SQL_TYPE_TIME</code>
<code>SQL_LONGVARCHAR</code>	<code>SQL_BIGINT</code>	<code>SQL_DOUBLE</code>
<code>SQL_TYPE_DATE</code>	<code>SQL_REAL</code>	<code>SQL_TINYINT</code>
<code>SQL_TYPE_TIMESTAMP</code>	<code>SQL_WVARCHAR</code>	<code>SQL_INTEGER</code>
<code>SQL_BIT</code>	<code>SQL_LONGWVARCHAR</code>	<code>SQL_DECIMAL</code>

Handling DSI_CATALOGONLY_METADATA

1. When called with `DSI_CATALOGONLY_METADATA`, `QSDATAENGINE::MakeNewMetadataTable()` will return an instance of `QSCATALOGONLYMETADATASOURCE()`.
2. You will need to change:
 - a. `QSSchemaOnlyMetadataSource::QSSchemaOnlyMetadataSource()`
The example constructor does nothing. You should modify this method to load the catalogs defined within your data store.
 - b. `QSCatalogOnlyMetadataSource::GetMetadata()`
In the SimbaEngine Quickstart Driver, this method



returns a single row. You should modify this method to retrieve catalog only information from your data store for the current row.

- c. `QSCatalogOnlyMetadataSource::Move()`
Since the SimbaEngine Quickstart Driver only contains a single catalog, this method returns false after fetching the first row. You will need to modify this method to iterate appropriately over your catalog information list.

Handling `DSI_TABLETYPEONLY_METADATA`

1. When called with `DSI_TABLETYPEONLY_METADATA`, `QSDataEngine::MakeNewMetadataTable()` will return an instance of `QSTableTypeOnlyMetadataSource()`.
2. The SimbaEngine Quickstart Driver exposes support for all table types: `TABLE`, `VIEW`, `SYSTEM TABLE`.
3. Modify `QSTableTypeOnlyMetadataSource::InitializeData()` if you don't support all of these table types.

Handling `DSI_TABLES_METADATA`

1. When called with `DSI_TABLES_METADATA`, `QSDataEngine::MakeNewMetadataTable()` will return an instance of `QSTablesMetadataSource`.
2. You will need to change:
 - a. `QSTablesMetadataSource::QSTablesMetadataSource ()`
In the SimbaEngine Quickstart Driver, the constructor seeds a list of tables through the use of the utility function `GetTables()`. This accesses the file system to return a list of defined files indexed by schema name. Since the SimbaEngine Quickstart Driver does not support schemas, there is a single "null" entry in the map associated with a vector of table names.

You should modify this constructor to initialize it correctly with a list of tables from your data store. For each table, you should provide the catalog name, schema name and table name.

- b. `QSTablesMetadataSource::GetMetadata()`
In the SimbaEngine Quickstart Driver, this method accesses the vector stored in the

current schema. Modify this method to retrieve correctly the table information from the structure you initialized in the previous step.

- c. `QSTablesMetadataSource::Move()`
In the SimbaEngine Quickstart Driver, this method iterates over the map initialized in the constructor as follows:

While there are schemas (keys) in the map

Iterate over the values vector associated with the schema.

Modify this method to iterate over the data structure you created in the constructor.

Handling `DSI_COLUMNS_METADATA`

1. When called with `DSI_COLUMNS_METADATA`, `QSDataEngine::MakeNewMetadataTable()` will return an instance of `QSColumnsMetadataSource`.
2. You will need to modify:
 - a. `QSColumnsMetadataSource::QSColumnsMetadataSource ()`
In the SimbaEngine Quickstart Driver, the constructor seeds a list of tables with a utility function called `GetTables()`. This accesses the file system to return a list of defined files indexed by schema name. Since SimbaEngine Quickstart Driver does not support schemas, there is a single "null" entry in the map associated with a vector of table names.

You must modify this constructor to initialize it correctly with a list of tables defined within your data store. For each table, you must provide the catalog name, schema name and table name, as well as the columns defined for the table.

- b. `QSColumnsMetadataSource::GetMetadata()`
In the SimbaEngine Quickstart Driver, this method accesses the vector stored in the current schema. Modify this method to correctly retrieve the table information from the data structure you initialized in the previous step.



- c. `QSColumnsMetadataSource::Move()`
Modify this method to iterate over the structure you created in the constructor.

Congratulations! You can now retrieve metadata from within your data store. You should be able to run `SQLTables()`, `SQLColumns()` and `SQLGetTypeInfo()` from within `ODBCTest32.exe` (Unicode) and see the correct metadata returned.

Day Four

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

TODO #8: Open A Table.

`QSDDataEngine::OpenTable()` is the entry point where Simba SQL Engine requests tables involved in the query be opened. You must modify this method to check that the supplied catalog, schema and table names are valid and correspond to a table defined in your data store. If not, you should return null to indicate that the table does not exist.

If the inputs are valid, a new instance of `QSTable` will be returned.

`QSTable` is an implementation of `DSIExtSimpleResultSet`, an abstract class provided by Simba that provides for basic forward-only result set traversal. The main role of `QSTable` is to translate the stored data from your native data format into SQL Data types.

We implemented the `SimbaEngine Quickstart Driver for Tabbed Unicode Files`. The `SimbaEngine Quickstart Driver` translates the text from UTF16-LE strings into the SQL Data types defined for each column within the configuration dialog.

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

- Return the catalog, schema and table names for your table:
 - `QSTable::QSTable()`
 - The constructor must be modified to take in the catalog, schema and table names and save them in member variables (`m_catalog`, `m_schema` and `m_table` respectively).
 - `QSTable::GetCatalogName()`: Returns `m_catalog`;
 - `QSTable::GetSchemaName()`: Returns `m_schema`;
 - `QSTable::GetTableName()`: Returns `m_table`;
- Return the columns defined for your table.
 - `QSTable::InitializeColumns()`: This method must be modified so that, for each column defined in the table, you define a `DSIResultSetColumn` in terms of SQL types.

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

```

SQLEngine::SEAutoPtr<DSIResultSetColumns>
columns;
    Get all the column information from your data
store for the table
For Each Defined Column
{
SQLEngine::SEAutoPtr<DSIColumnMetadata>
columnMetadata(
    new DSIColumnMetadata());
columnMetadata->m_catalogName = m_tableName;
columnMetadata->m_schemaName = m_schemaName;
columnMetadata->m_tableName = m_tableName;
columnMetadata->m_name = //column name
columnMetadata->m_label = //localized column name
columnMetadata->m_unnamed = false;

columnMetadata->m_charOrBinarySize = //the length in
bytes of the column

columnMetadata->m_nullable = DSI_NULLABLE;

// Change the first parameter of this method to the SQL
// Type that maps to your data store type.

```



```
SqlTypeMetadata* sqlTypeMetadata =
    SqlTypeMetadataFactory::MakeNewSqlTypeMetadata
    (SQL_WVARCHAR,
     TDW_BUFFER_OWNED);

columns->AddColumn(
    new DSIResultSetColumn( sqlTypeMetadata,
    columnMetadata.Detach())
    );
}

m_columns.Attach(columns.Detach());
```

- Data Retrieval
 - QSTable::MoveToBeforeFirstRow()
 - QSTable::MoveToNextRow()
 - QSTable::GetData()

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

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

In the SimbaEngine Quickstart Driver, QSTable uses a TabbedUnicodeFileReader that provides an interface to navigate between lines within a Unicode text file. This class preprocesses each row in the file to determine the starting file offset of each column in the row. Its GetData method takes a columnIndex and uses it to calculate the exact position in the file where the column's data resides. The method repositions the file and retrieves the data as if from a byte-buffer. See Section 4.6, Technical Note: Data Retrieval, for a brief overview of data retrieval.

- QSTable::DoCloseCursor()

This is a callback method called from Simba SQL Engine to indicate that data retrieval has completed and that you may now do any tasks related to closing the connection to your data store. Once positioned on a particular row, Simba SQL Engine will call this method once for each column in the table that is involved in the query.

Congratulations! You can now retrieve data from your data store. You should be able to execute queries from any ODBC-enabled application (e.g.: Microsoft Excel, Microsoft Access, Microsoft SQL Server, Business Objects Crystal Reports) and see the results returned from your data store.

Day Five

Today's goal is to start productizing your driver.

TODO #9: Update Messages xml file name.

All the error messages used within your DSI implementation are stored in a file called QSMessages.xml. Rename this file to something appropriate to your data store and update the line associated with the TODO to match.

You should also open the error messages file and change all instances of the following items:

1. The letters "QS" to a two letter abbreviation of your choice
2. The word "Quickstart" to a name relating to your driver

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 Quickstart Driver for your organization.

TODO #10: Update Driver name.

TODO #11: Update Driver name.



There are two error messages returned by the driver in the event that it can't find the messages XML file. These are hard coded into the DSI implementation and are identified by TODO 10 & 11. Simply change "[Quickstart]" to "[name of your driver]" – note that you must keep the square brackets.

You are now done with all 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 #9.
2. Modify the build settings to rename the resulting DLL. You will need to modify the registry entry for the SimbaEngine Quickstart Driver ODBC DSN to point to the renamed driver, or create a new registry entry, to be able to continue to use the renamed driver.
3. 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. You can find this application labeled "Data Sources (ODBC)" in Start->Settings->Control Panel->Administrative Tools. The SimbaEngine Quickstart Driver project file will also create an example ODBC configuration dialog for you. You can find the source under the Setup folder within the SimbaEngine Quickstart Driver project.

Technical Note: 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. QSTable::GetData()

Both methods will provide a way to uniquely identify a column within the current row. For MetadataSource, the Simba SQL Engine will pass in a unique column tag (see DSIOutputMetadataColumnTag.h). For QSTable, the Simba SQL Engine 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 your cell's value. This class is a wrapper around a buffer managed by the Simba SQL Engine.

To access the buffer, you simply 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 QSTable::InitializeColumns() to SQL_INTEGER, then when QSTable::GetData() is called for column 1, you will be passed a SQLData that wraps an int data type. For MetadataSources, 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 QSTypeUtilities.h for an example on how to handle character or binary data.

2. in_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. in_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.



Technical Note: How to Add Schema Support

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

Handling DSI_SCHEMAONLY_METADATA

1. `QSCONNECTION::SetConnectionPropertyValues()` currently returns 0 (false) for the value of `DSI_CONN_SCHEMA_USAGE`, indicating that the driver does not support schemas. Set this value to 1 (true) to add schema support to your driver.
2. `QSDATAENGINE::MakeNewMetadataTable()` throws an exception when called with `DSI_SCHEMAONLY_METADATA` because schemas are not currently supported. If your data store supports schemas, uncomment the region of code that handles `DSI_SCHEMAONLY_METADATA`. When uncommented, the code will return an instance of `QSSchemaOnlyMetadataSource`.
3. You will need to change:
 - a. `QSSchemaOnlyMetadataSource::QSSchemaOnlyMetadataSource()`
In the SimbaEngine Quickstart Driver the constructor seeds a predefined list of schemas within the `m_schemas` member variable. You should modify this to load the schemas defined within your data store.
 - b. `QSSchemaOnlyMetadataSource::GetMetadata()`
In the SimbaEngine Quickstart Driver, this method accesses the `std::vector` initialized in the constructor using the current row number as the index. Modify this method to retrieve schema information from your data store for the current row.
 - c. `QSSchemaOnlyMetadataSource::Move()`
In the SimbaEngine Quickstart Driver, this method uses an integer to iterate over the `m_schemas` member variable. This method will return false when the current row number is larger than `m_schemas.size()`. Modify this method to iterate over the schema information from your data store.

`schemas.size()`. Modify this method to iterate over the schema information from your data store.

Specifications

SimbaEngine SDK supports the following platforms and tools:

Stand-alone Data Driver and Server Platforms – All SimbaEngine SDK components support the 32- and 64-bit versions of Windows, UNIX and Linux.

Client Platforms – SimbaClient for ODBC supports the 32- and 64-bit versions of Windows, UNIX and Linux. SimbaClient for JDBC supports all client operating systems running a Java Virtual Machine (JVM).

Development Software – **Windows:** Microsoft Visual Studio.NET 2008, Microsoft Visual Studio.NET 2005. **UNIX:** For most platforms the native compiler is supported. **Linux:** GNU Compiler environment. ODBC.NET Provider required for ADO.NET functionality. OLE DB / ODBC Bridge required for OLE DB functionality.

Desktop Tool Support – ODBC capable applications, including Microsoft Office and SAP Business Objects Crystal Reports.



About Simba Technologies Inc.

Simba Technologies Inc. is the recognized world leader in standards-based data access products and solutions. We work with the world's leading software companies to deliver first class data connectivity solutions.

Simba is a pioneer in ODBC, MDX, OLE DB for OLAP (ODBO) and XML for Analysis (XMLA). Since 1991, we've developed advanced data access solutions for thousands of end users. Today, more than half of all MDX providers have been built with Simba technology, and through a partnership with Microsoft, Simba's SQL technology has been installed on more than 30 million desktops worldwide.

Our firm commitment to delivering the highest customer value through innovative solutions and expert support has gained us a reputation as the industry leader for data connectivity solutions.

©2009 Simba Technologies Inc. All Rights Reserved.
Printed in Canada.

Simba Technologies Incorporated

938 West 8th Avenue
Vancouver, BC Canada
V5Z 1E5

Tel. +1.604.633.0008
Fax. +1.604.633.0004
Email. solutions (at) simba.com
www.simba.com

Simba and the Simba logo are trademarks of Simba Technologies Inc. All other trademarks or service marks are the property of their respective owners.