



SimbaEngine SDK 8

Getting Started and SDK User's Guide

For version 8.0.2

Simba Technologies Inc.



Copyright ©2009–2010 Simba Technologies Inc. All Rights Reserved.

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this publication, or the software it describes, may be reproduced, transmitted, transcribed, stored in a retrieval system, decompiled, disassembled, reverse-engineered, or translated into any language in any form by any means for any purpose without the express written permission of Simba Technologies Inc.

Simba Trademarks

Simba, the Simba logo, SimbaEngine, SimbaEngine C/S, SimbaClient, SimbaD20, SimbaEngine SDK and SimbaODBC are registered trademarks of Simba Technologies Inc. All other trademarks and/or servicemarks are the property of their respective owners.

Simba Technologies Inc.

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

Tel. +1.604.633.0008
Fax. +1.604.633.0004

www.simba.com

Printed in Canada

Third Party Trademarks

Copyright © 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.

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.

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.

Table of Contents

- 1 Welcome 1
- 2 Introducing SimbaEngine SDK 2
 - 2.1 A Simple ODBC Driver 2
 - 2.2 An ODBC Driver with an SQL Engine..... 3
 - 2.3 Mix and Match the Components..... 4
 - 2.4 How Do The Simba Components Work Together? 5
 - 2.5 How Do I Use the SDK? 8
 - 2.6 What’s Next? 11
- 3 Using SimbaEngine SDK 13
 - 3.1 How the Components Work Together 13
 - 3.2 How to Use the SDK 15
 - 3.3 How does each component work? 21
- 4 Build an ODBC Driver in Five Days 29
 - 4.1 Day One 31
 - 4.2 Day Two..... 33
 - 4.3 Day Three..... 35
 - 4.4 Day Four 36
 - 4.5 Day Five 38
 - 4.6 Technical Note: Data Retrieval 39
 - 4.7 Technical Note: How to Add Schema Support..... 40
- 5 Frequently Asked Questions..... 42
- Appendix A: Platforms and System Requirements 46
- Appendix B: Installation 47
- Appendix C: Windows 32-Bit vs. 64-Bit 48
- Appendix D: How Do I Get Support? 50

Table of Figures

Figure 1: The architecture of an ODBC driver built with SimbaODBC..... 3

Figure 2: The architecture of an ODBC driver built with SimbaEngine. 4

Figure 3: Three examples of SimbaEngine SDK architecture for comparison. 5

Figure 4: The Simba Client/Server architecture showing the SimbaClient for ODBC connecting remotely to Simba Server. 7

Figure 5: A comparison of three different data access standard clients connecting remotely to Simba Server. 8

Figure 6: An imaginary non-tabular database schema is represented by tables and columns containing the same data and linked in the same way. The two database are equivalent, but Simba SQL Engine can directly address the tabular database. 10

Figure 7: Three examples of system stacks you can build with SimbaEngine SDK. 14

Figure 8: The structure of the folders installed by SimbaEngine SDK. 16

Figure 9: The DSNs installed with SimbaEngine SDK or available in .INI files..... 17

Figure 10: A look inside Simba SQL Engine showing the Preparation and Execution components communicating via the Execution Tree..... 22

Figure 11: The progression of creating an AE-Tree, passing down a filter to the data store, and replacing the original table node with a new filtered table node. 24

Figure 12: The architecture of Simba Client/Server showing the Client/Server protocol connecting the components. 25

Figure 13: A high-level view of the generally-successful design pattern for a DSI implementation driver. 30

1 Welcome

Welcome to SimbaEngine SDK, the quickest and easiest way to build an ODBC 3.52, JDBC or ADO.NET database driver. SimbaEngine SDK helps you to build data access solutions for any data store whether it is SQL-capable or not. Using SimbaEngine SDK, you can build standards-based data drivers for mainstream SQL databases and for non-standard, non-SQL databases such as Object Oriented, Hierarchical, SCADA, Key-Value, Web Service and ISAM data stores. In fact, if you can make your data store look like tables and columns, you can build a SimbaEngine SDK data driver for it and your customers can query your data using Microsoft Excel and Crystal Reports.

Collaborative Query Execution

With the unique Collaborative Query Execution technology in SimbaEngine SDK, you can have your execution engine take over execution of any part of a query. For remote data stores this speeds up query processing by reducing the amount of data that Simba SQL Engine must retrieve. For high-performance data stores, like column-oriented databases or key-value stores, this allows you to expose the high-performance of your data engine while Simba SQL Engine only executes the remaining portions of the query. This unique capability allows Simba SQL Engine and your high-performance data store work collaboratively to deliver the highest performance to your customers.

Platform Support

SimbaEngine SDK is available on a wide range of platforms including Windows, Linux, Solaris, HP-UX and AIX, in both 32-bit and 64-bit versions. Using SimbaClient and SimbaServer, you can use a standards-based data driver to access your data on any platform, from any platform, regardless of processor architecture or word length. Whether your data is strictly ANSI or you have customers all around the World, SimbaEngine SDK can handle it with Unicode capability designed-in. Your new standards-based data driver will be ready to localize anywhere.

Data Access Standards

You can build remote and local ODBC 3.52, JDBC 3.0 and ADO.NET drivers using a single Data Store Interface (DSI) implementation to connect SimbaEngine SDK to your data store. This lowers your development and testing costs and allows you to focus on delivering value to your customers. A simple, iterative, development approach allows you to quickly deliver standards-based data access to your customers, and later add performance and functional improvements, as they demand them. You can strategically assign your development resources to features that your customers want.

Where do I go next?

For more information on how you can use SimbaEngine SDK, keep reading the next section, Introducing SimbaEngine SDK. For specific information about platforms supported and system requirements, please see Appendix A: Platforms and System Requirements. To get started prototyping a driver for your data store, please read section Build an ODBC Driver in Five Days.

2 Introducing SimbaEngine SDK

SimbaEngine SDK enables you to create data access solutions to any data store – including those that can process SQL and those that do not understand SQL at all, such as object-oriented, ISAM and SCADA data sources. In fact, if you can make your data store look like it uses tables and columns, SimbaEngine SDK can create a driver for it that your customers can use with common reporting applications.

Solutions created with SimbaEngine SDK have a clear, simple structure that makes them easy to understand. This section introduces the structure of the solutions you create with SimbaEngine SDK and explains how the different parts work together. As you continue to learn about SimbaEngine SDK, you can relate your newfound knowledge to your understanding of the standard structure. Note that in the diagrams of SimbaEngine SDK solutions below, program control flows downward and retrieved data flows upward.

SimbaEngine SDK is a collection of database access tools packaged into a small set of components. The components fit together in different ways to solve a variety of data access problems. Included in SimbaEngine SDK are components based on widely used and supported data access standards like ODBC, JDBC, and SQL. This means that the data access solutions you create can be used by common reporting applications like Crystal Reports and Microsoft Excel, as well as a wide variety of enterprise, specialized and custom applications.

2.1 A Simple ODBC Driver

Building a driver for a SQL-capable data store

Figure 1, below, is a diagram of an ODBC driver created with SimbaEngine SDK to connect to an SQL data store. In this case, the Customer Data Store understands SQL-92, or a variant. The application creates SQL queries and sends them to the ODBC driver, where they are possibly modified and sent on to the data store. The data store executes the SQL queries and creates a result set. The ODBC driver can then move the result set from the data store back to the application.

There are several things to note in Figure 1. First is the layered, component nature of the system, with well-defined interfaces between the components. The ODBC API is the standard interface used by most data access applications, such as Microsoft Excel and Crystal Reports, to access relational data stores. The ODBC API hides the idiosyncrasies of the data store from the application so the application can be written once and be able to connect to many different data stores. The ODBC Driver Manager is a shared library (.DLL, .SO) that mediates between the application and the ODBC driver. It performs error checking and translation that widens the range of the ODBC drivers that can be used by the application. Microsoft supplies the Driver Manager on Windows. Simba supplies a driver manager for non-Windows operating systems.

In Figure 1 the two components separated by the DSI API make up the ODBC driver itself. The majority of the functionality is contained in the SimbaODBC component. This component implements all the functionality of ODBC so that you don't have to. This includes error checking; driver, session and statement management; and keeping track of the ODBC details expected by the Driver Manager and the application. Simba supports and maintains the SimbaODBC component as part of SimbaEngine SDK and makes sure that you don't have to worry about any changes to the ODBC API or the way applications use it.

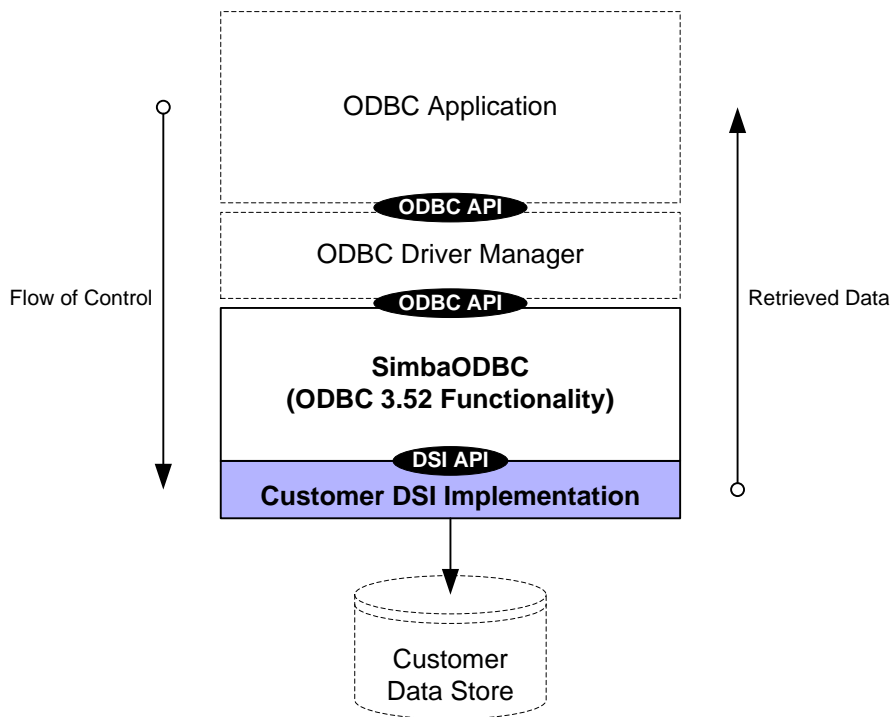


Figure 1: The architecture of an ODBC driver built with SimbaODBC.

The SimbaODBC component communicates with the Customer DSI implementation via the Data Store Interface, or DSI. This interface is common to all SimbaEngine SDK components that communicate with customer code. The Customer DSI implementation is the component that connects directly to the data store and it is specific to each different data store. It is custom designed for each different data store and its interface.

2.2 An ODBC Driver with an SQL Engine

Building a driver for a non-SQL data store

Many data stores do not understand SQL. In fact, many data stores are not organized as tables and columns at all. For example, object-oriented, network and SCADA data stores have non-tabular storage, and a database system that allows users to query familiar business objects, such as invoices and shipments, may not store its data as those entities at all. In these cases you must add an SQL engine to the system to enable ODBC applications to access the data store. Figure 2, below, is a diagram of this kind of ODBC driver.

In Figure 2 you can see the familiar SimbaODBC component and the DSI API, but there is a new component inserted between them – Simba SQL Engine – that provides the SQL-92 processing required for ODBC and other standard interfaces. The DSI API remains the same and performs the same service in this case as it did in Figure 1, but you can now use common reporting applications to access non-SQL data stores as well.

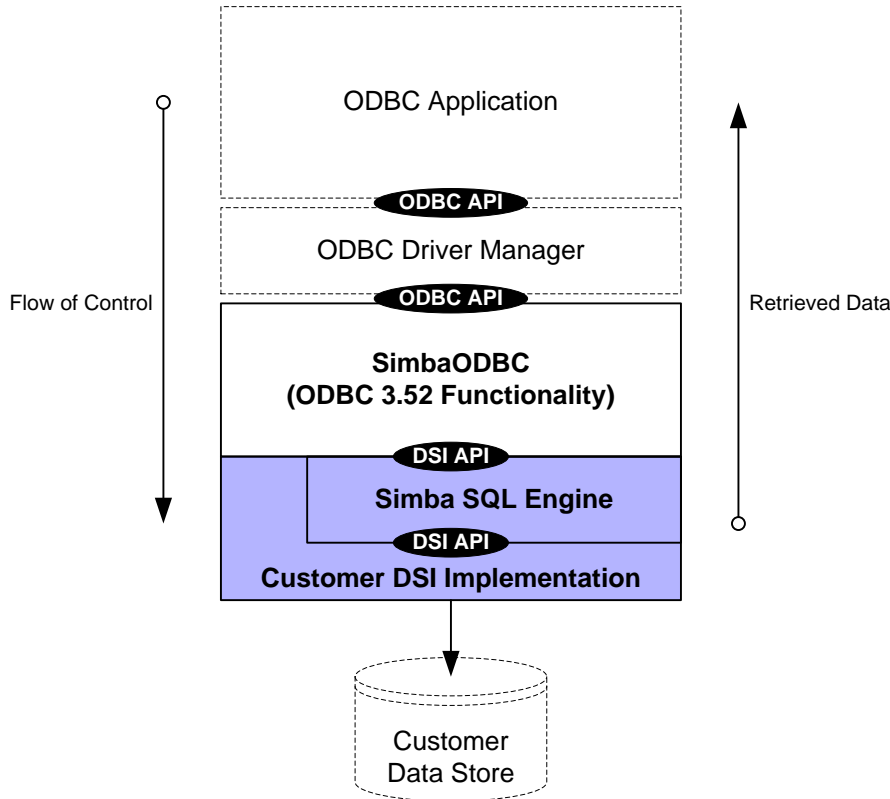


Figure 2: The architecture of an ODBC driver built with SimbaEngine.

The key idea is that SimbaEngine SDK provides access to both SQL and non-SQL data stores using its simple layered architecture. This architecture expands to cover remote data access (client/server) whether supplied by SimbaEngine SDK or the customer data store, many industry standard data access interfaces, and data stores available through managed languages such as Java and C#. The architecture remains the same and the DSI API provides the data store abstraction to allow many different solutions with one DSI implementation.

2.3 Mix and Match the Components

One DSI implementation yields many solutions

Figure 3, below, shows three different solutions from SimbaEngine SDK. Note that even though the issues solved are very different, the architecture of the solutions remain the same. Example A is familiar, with SimbaODBC and the Customer DSI implementation combining to provide ODBC access to a local, SQL-capable data store. Example B shows the architecture from Figure 2 but this time working with a remote interface to the data store. This changes

the constraints placed on the Customer DSI implementation because network latency can create bottlenecks, but the architecture remains the same. Example C shows the Simba Client/Server components in use to deliver standard data access to remote applications. However, the same architecture applies and the same Customer DSI implementation created for Example B can be used.

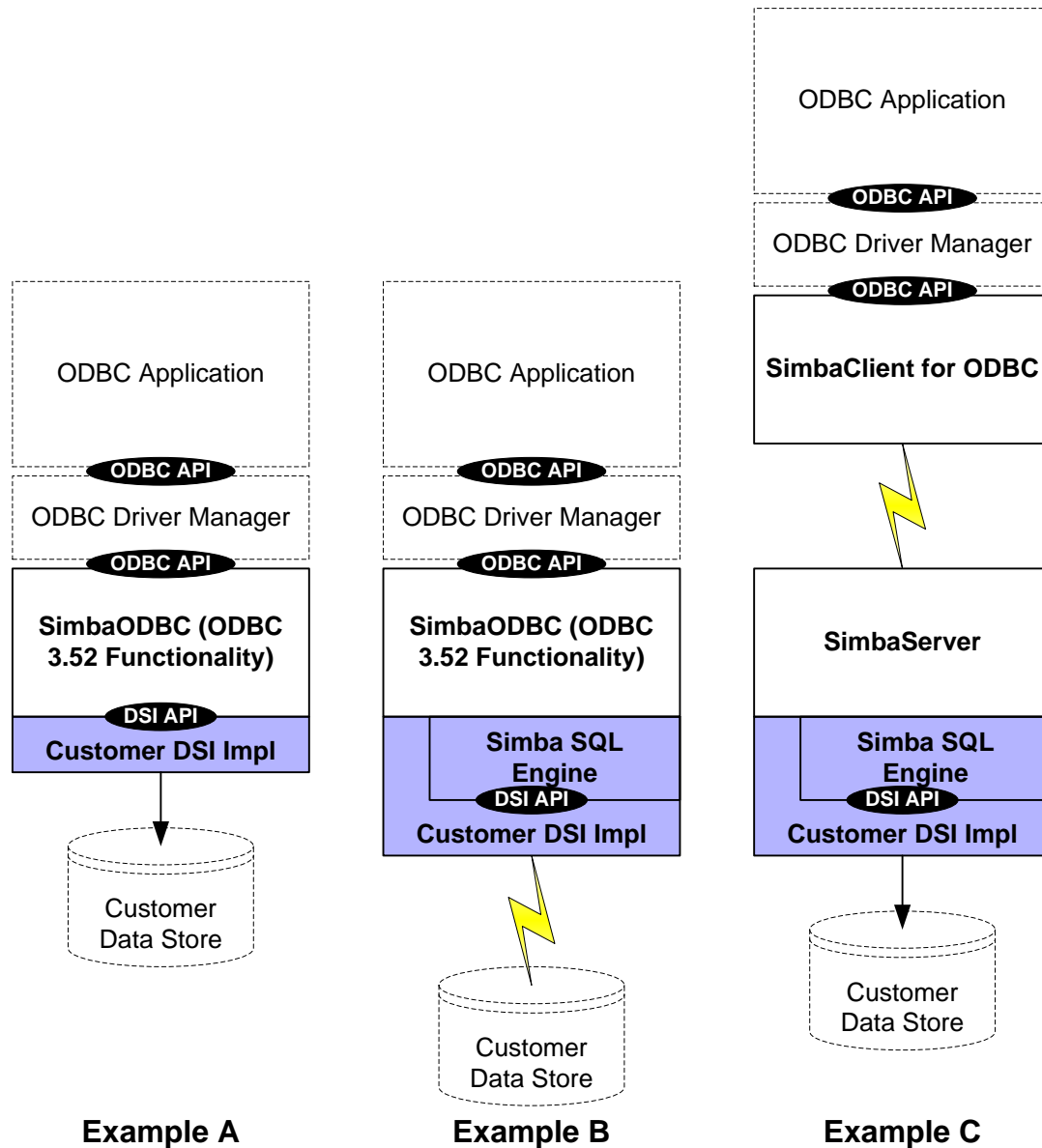


Figure 3: Three examples of SimbaEngine SDK architecture for comparison.

2.4 How Do The Simba Components Work Together?

How do they communicate?

The real secret of the components in SimbaEngine SDK is that they can all connect together using two simple interfaces. We have designed the interfaces so they do exactly what is

required, which allows them to be simple and efficient. Using only a few interfaces means more flexibility in the way they can connect together. The following subsections describe this.

2.4.1 The Data Store Interface (DSI)

A canonical database access interface

The DSI API defines a generic view of an SQL database that is independent of data access standards such as ODBC, JDBC and ADO.NET. It is object-oriented and simple to use. The ODBC, JDBC and ADO.NET interfaces can translate to the DSI API, and this is exactly what SimbaODBC does. It translates the ODBC interface to the DSI interface. The DSI API is easier to use than the ODBC interface so it is easier to write a DSI implementation that will translate to a custom data store.

The DSI interface serves as a common API. SimbaEngine SDK includes code to map from standard data access interfaces to the DSI. If you create code to map from your data store to the DSI, you create drivers to each of the standard interfaces. SimbaEngine SDK actually makes JDBC and ADO.NET available via its Client/Server component, but the principle remains. You can use one implementation of the DSI interface in several ways. You can see this in Figure 3, above, in which each different architecture illustrated has the DSI interface and a DSI implementation at the bottom of its stack.

To learn more about the DSI please read the **SimbaEngine User's Guide**. For detailed information about the DSI API method calls, please see the on-line documentation found in the Documentation folder in the installed SimbaEngine SDK.

2.4.2 The Client/Server Protocol

Another simple interface used by SimbaEngine SDK is the Simba Client/Server protocol. The Simba Client/Server protocol is a network protocol that works on any network to provide remote access to a DSI implementation. Figure 4, below, shows the architecture of the stack created with the Simba ODBC Client, Simba Server and a DSI implementation. Simba Client communicates with Simba Server using the Simba Client/Server protocol. Simba Server translates the Simba Client/Server protocol to the DSI interface, and any DSI implementation can be linked to the Simba Server to provide remote data access.

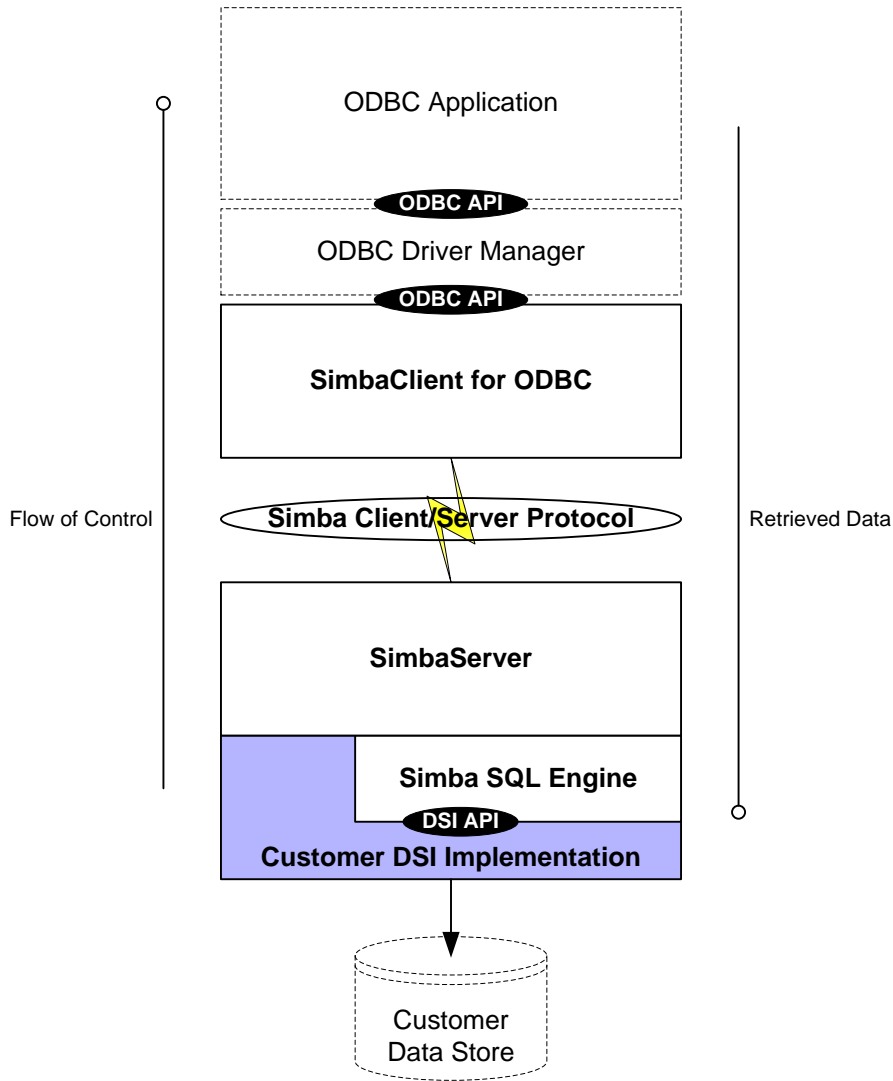


Figure 4: The Simba Client/Server architecture showing the SimbaClient for ODBC connecting remotely to Simba Server.

As an aside, SimbaClient uses the DSI interface internally. SimbaClient is the combination of SimbaODBC linked to a DSI implementation that translates the DSI interface to the Simba Client/Server protocol.

Figure 5, below, shows the architecture of three different stacks that each start with a different type of application; ODBC, ADO.NET and Java. Each client translates its specific data access interface to the Simba Client/Server protocol for transmission to the server.

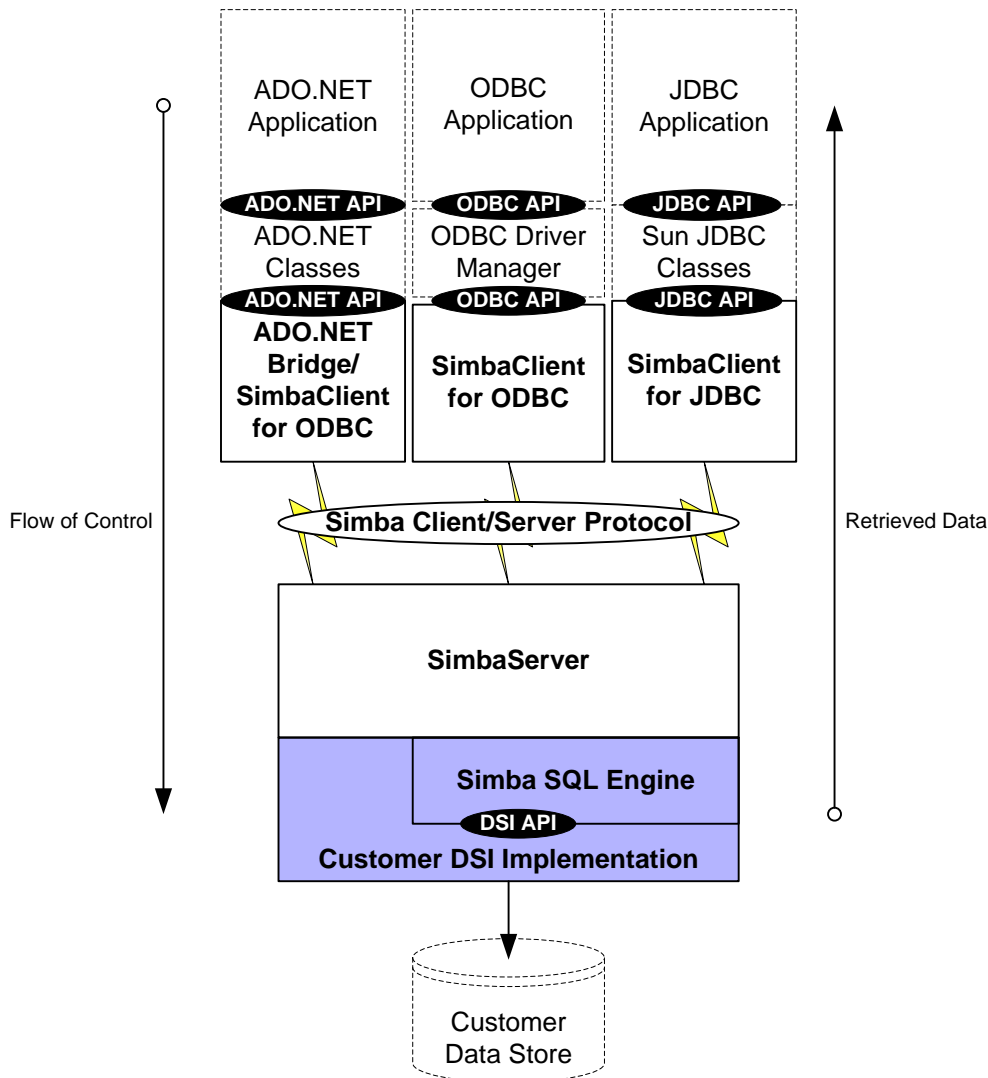


Figure 5: A comparison of three different data access standard clients connecting remotely to Simba Server.

2.5 How Do I Use the SDK?

How you use SimbaEngine SDK depends on what you want to do, but there are a few common, simple steps. There are essentially two ways to approach SimbaEngine SDK. One way is quickly to build a proof-of-concept driver in a few days that will convince you that it is possible and help you to learn about SimbaEngine SDK. The other is to build a complete

commercial driver for your data source that you will ship to your customers. In both cases, the steps are similar:

1. Plan how to translate your data store schema to the DSI view of the world.
2. Make a copy of the appropriate Quickstart driver folders and rename them.
3. Implement your plan for translating your data store to the DSI.

Of course, your planning and implementing activities will take much longer for a commercial driver than for a proof-of-concept. However, the difference is largely the amount of detail required. In both cases, the goal is to arrive at a working executable that is either an ODBC driver or a server that you can test with a common reporting tool.

How can I get started on a real driver?

If you want to start prototyping a driver right away to make sure that SimbaEngine SDK can access your data store, jump to Section 4, Build an ODBC Driver in Five Days. If you already know that SimbaEngine SDK works for your data store you can continue reading this section, and then continue with Section 3, Using SimbaEngine SDK, to learn more about preparing to design and build a commercial data access solution.

2.5.1 Database Schema Translation

How do I make my data store work with SimbaEngine SDK?

The first step is to plan how you will translate your data store schema to the DSI view of the world. The DSI represents the data store as a series of tables and columns, and the purpose of the DSI implementation is to translate the real data store schema into the DSI representation. If you can imagine this translation then you can make SimbaEngine SDK work for you.

Figure 6, below, illustrates visually the kind of translation required. An existing database uses an object oriented or networked schema to store the data because it suits the primary purpose of the database. However, a relational SQL execution engine cannot directly use this schema. If you represent the same database as tables and columns, even though you do not actually transform the database into this new form, it fits the relational paradigm. Now you can write a DSI implementation to create this view of the data and SimbaEngine can execute SQL queries against it. In this way, any database you can represent as tables and columns can be accessed by SimbaEngine and made accessible to common reporting tools.

Translating a Network Database into Tables and Rows

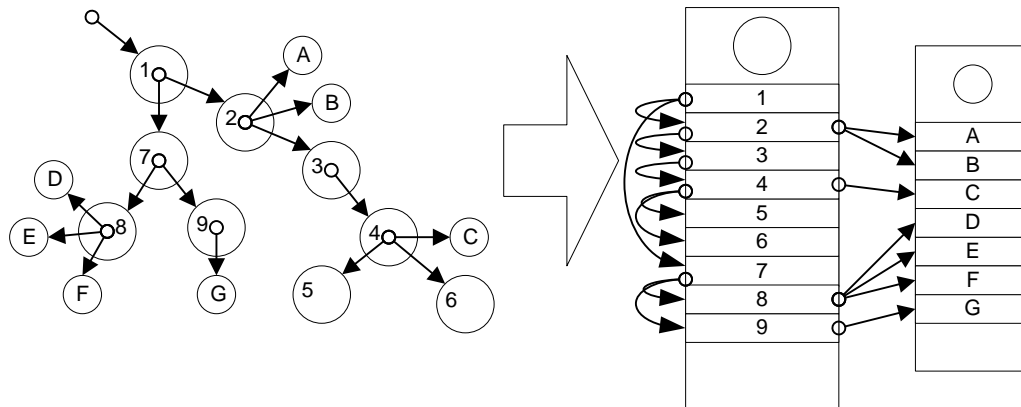


Figure 6: An imaginary non-tabular database schema is represented by tables and columns containing the same data and linked in the same way. The two database are equivalent, but Simba SQL Engine can directly address the tabular database.

A simple approach you can use to create a tabular view of your data store is to read the entire database into temporary tables. You can then access the temporary tables through the DSI. However, this is inefficient and only works for very small databases. A more efficient method is to create virtual tables and then access the original database when SimbaEngine requests data through the DSI interface. There are many approaches and the one you choose must suit your data store.

2.5.2 Preparing Your Workspace

When you know how to translate your data store schema to the DSI view of the world, the next step is to create a workspace for yourself where you can begin work on your new DSI implementation. To work on a new DSI for SimbaEngine, copy the entire folder called “SimbaEngineQuickstart” and rename it to something appropriate. Contained in this folder is the source code and the project files or make files to build a complete, working SimbaEngine Quickstart sample DSI implementation that will read text files. Your job is to transform this source code so your data store supplies the data instead of the text files.

Also included is source code for an ODBC configuration DLL you can modify. This configuration DLL allows your customers to create and modify ODBC DSNs that use your new ODBC driver to connect to your data store.

2.5.3 Creating a Database Access Solution

What does my driver look like when it is done?

The goal is to compile and link your DSI implementation into a DLL or shared object that common reporting tools can use to access your database. In the process, the project files or make files will link in the appropriate SimbaODBC and SimbaEngine libraries to complete the

driver. If you are building a client/server solution, the project files or make files will also link in the appropriate Simba Client/Server libraries to complete the server executable. In all cases, the goal is to create an executable file that can be accessed by common reporting tools and will access your data store when SimbaEngine executes an SQL statement.

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 interface.

2.6 What's Next?

If you have read this section, you have a high-level understanding of how SimbaEngine SDK works and how you can combine its components to provide access to your data store in a variety of environments. Your next step is to decide what you need to do next – start prototyping right away, learn more about SimbaEngine SDK, or dig deeper into the various tools.

2.6.1 Prototype Data Retrieval from your Database

If you want to start exploring SimbaEngine SDK and investigating how you can access your data store with SimbaEngine SDK, you can jump to Section 4, Build an ODBC Driver in Five Days. This gives you step-by-step instructions to create a working driver using the SimbaEngine Quickstart sample driver. It is an excellent way to learn about SimbaEngine SDK while exploring what you need to do to connect it to your data store.

2.6.2 Learn More about the SDK

If you have already prototyped a driver for your data store, or if you want to learn more to help you plan a full commercial driver, continue reading Section 3, Using SimbaEngine SDK. This section goes into more detail about the internal workings of SimbaEngine SDK and its components. You must consider many things when building a commercial driver such as data types, indexes, caching and pass-down optimizations. They are all easier to understand with a better understanding of the architecture of SimbaEngine SDK.

2.6.3 Learning More about the Architecture and Theory

The components included in SimbaEngine SDK, such as SimbaODBC and Simba SQL Engine, contain sophisticated data access tools that perform a lot of complicated work to make it easy for you to access your data store. You will be able to design and build a better DSI implementation if you understand the theory of how these data access tools work and how they use your code.

To learn more about ODBC and building drivers for SQL-enabled data stores, please read the SimbaEngine User's Guide. This describes how SimbaODBC handles application requests

through its ODBC interface and how it requests information and data through the DSI. The on-line DSI documentation provides specific details about each DSI method.

To learn more about Simba SQL Engine and building drivers for non-SQL capable data stores, please read the SimbaEngine User's Guide. This describes how Simba SQL Engine transforms SQL-92 statements into executable statements, how the statements are executed using the DSI method calls, and how the queries are optimized to provide the best performance to your customers. **NOTE: THE SimbaEngine User's Guide IS NOT YET PUBLISHED.**

3 Using SimbaEngine SDK

Writing code, building drivers.

We have designed SimbaEngine SDK to help you create data access solutions so that common reporting tools, like Crystal Reports and Microsoft Excel, can query your data store. If your data store does not understand SQL then the solution will include Simba SQL Engine to process SQL queries. If your data store understands SQL then Simba SQL Engine will not be included and SimbaEngine SDK tools will pass on the SQL queries. SimbaEngine SDK provides the components to create a front end to your data store that includes a complete implementation of ODBC 3.52, JDBC 3.0 or ADO.NET, as well as a Client/Server component that allows you to create a remote solution.

This section describes how the parts of SimbaEngine SDK work together to help you create the data access solution you need. It explains how we have structured SimbaEngine SDK when installed, how the components fit together in use, how to build your solution, and how to approach designing and implementing your solution. At the end of this section, you should know what your next steps should be, whether you are evaluating SimbaEngine SDK for possible inclusion in your product or designing a complete, optimized commercial database access solution. When you are done, there are pointers to help you find detailed information about the various components found in SimbaEngine SDK.

3.1 How the Components Work Together

Components + Interfaces = Executables

The goal of SimbaEngine SDK is to create an executable file that will run and enable access to your data store. This can be a Windows DLL, a Linux or UNIX shared object, a stand-alone server, or some other form of executable. You create the executable by linking libraries from SimbaEngine SDK with a DSI implementation you have written for your data store. What kind of executable you create depends on the solution you need, but you can use your DSI implementation in different ways. You can create many different solutions, but you only need to create one DSI implementation for your data store.

SimbaEngine SDK components and your DSI implementation link together through internal interfaces. Together they present external, standard interfaces for common applications to use. The DSI API and the Client/Server protocol are internal interfaces used to link components together into a single solution. The DSI API links your DSI implementation to SimbaEngine SDK components. This is usually a static link but it can be a dynamic link if required. The Client/Server protocol is an internal interface connected at run time when a SimbaClient wants to connect to a SimbaServer. All SimbaClients connect and communicate with SimbaServers using the same Client/Server protocol.

SimbaEngine SDK uses third-party components to perform common functions. Two of these are International Components for Unicode (ICU) and OpenSSL. SimbaEngine SDK uses ICU to convert between ANSI and Unicode representations of text, and between the various Unicode encodings. SimbaEngine SDK components call ICU functionality using ICU's own interface. The component is dynamically loaded on Windows, Linux and UNIX. OpenSSL is an implementation of the Secure Sockets Layer and the Transport Layer Security protocols as well as a full-strength cryptography library. SimbaEngine SDK uses OpenSSL to secure the network connection between SimbaClient and SimbaServer.

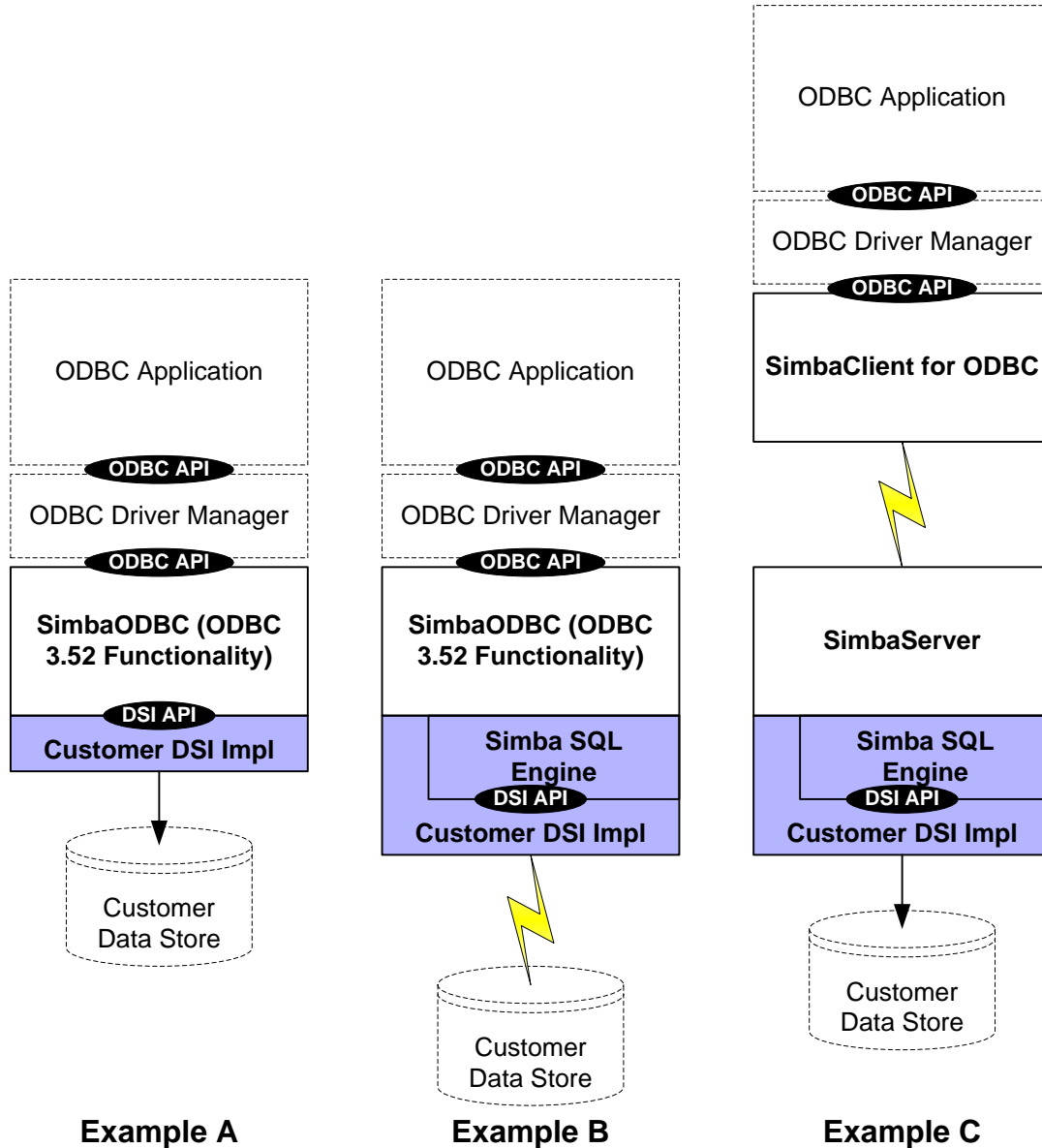


Figure 7: Three examples of system stacks you can build with SimbaEngine SDK.

Figure 7 illustrates three possible ways you can combine SimbaEngine SDK components to create different solutions for different situations. The completed data access solution presents a standard external interface to applications, such as ODBC, JDBC and ADO.NET.

Connecting from applications, configuring a data source

Standards-based applications connect to these interfaces at run time, and the connections are sensitive to proper configuration. Sometimes, as in the case of a JDBC or ADO.NET connection, the application itself handles the connection configuration via information handed to the standard interface at connect time. In the case of ODBC, a Data Source Name, or DSN, maintains the configuration information, and the application refers to the DSN at connect time. To change the connection configuration in the DSN, the user runs the Windows Data Source Administrator to execute a special configuration DLL included with each ODBC driver. With the configuration DLL, the user can change the configuration of a DSN within the specifications of the driver. Incorrectly configured DSNs are frustrating. Some ODBC configuration DLLs include a connection test so you can check the configuration immediately.

3.2 How to Use the SDK

The result of building a data access solution with SimbaEngine SDK is a DLL on Windows, or a shared object on Linux or UNIX, or a stand-alone executable server. In each case, the executable is the result of compiling custom DSI implementation code that connects to your data store, and then linking it with SimbaEngine SDK components. In the installed SimbaEngine SDK, you can find the libraries containing the data access components you need in the "DataAccessComponents" folder. In the "Examples" folder are sub-folders containing the working DSI source code and project files or make files to create different data access solutions. Ready-to-use versions of the example ODBC drivers and the server are available in the "Bin" folder for immediate use and reference.

The SimbaEngine SDK folders after installation

```

[PROGRAM FILES FOLDER]\Simba
  SimbaEngineSDK
    8.0
      Bin
      DataAccessComponents
      ErrorMessage
      Include
        DSI
        Product
        SQLEngine
        Support
      Lib
        win32
        x64
      ThirdParty
      Databases
      Codebase
      Text
      Documentation
      Setup
      SSLCertificates

[PERSONAL FOLDER]\SimbaEngine 8.0 Projects
  Examples
    SimbaEngineCodebase
    SimbaEngineQuickstart
  
```

NOTE:

[PROGRAM FILES FOLDER] is the default installation directory. On 32-bit Windows platforms, this folder is C:\Program Files. On 64-bit Windows platforms, this folder is C:\Program Files (x86).

[PERSONAL FOLDER] is the installation directory for SimbaEngine 8.0 Projects. On Windows versions older than Windows Vista, this folder is C:\Documents and Settings\

Figure 8: The structure of the folders installed by SimbaEngine SDK.

Where is SimbaEngine SDK installed on my machine?

When you install SimbaEngine SDK on Windows, it creates a series of ODBC DSNs. On Linux and UNIX there are sample ODBC.INI and ODBCINST.INI files from which you can copy driver and DSN information to modify your existing ODBC.INI and ODBCINST.INI. If you do not already have an existing ODBC.INI and ODBCINST.INI, you can use the ones included in SimbaEngine SDK directly by copying them to your \$HOME directory. The installed DSNs point to the Codebase and text file databases and to the ODBC drivers in the “Bin” folder. They have self-explanatory names; for example, the DSN called “CodebaseDSII” loads the SimbaEngine Codebase reference driver in the “Bin” folder and uses the Codebase database. The reference DSNs are immediately useable after installation so you can begin exploring right away. You can create more DSNs that point to your new driver by modifying and running the .reg file found in each Examples folder to install your new driver, and then using the ODBC Data Source Administrator to create the new DSNs. On Linux and UNIX, you can modify the ODBC.INI directly with a text editor.

DSN Name	Driver Location	Database Location
CodebaseDSII	..\Simba\SimbaEngineSDK\8.0\Bin	..\Simba\SimbaEngineSDK\8.0\Databases\Codebase
QuickstartDSII	..\Simba\SimbaEngineSDK\8.0\Bin	..\Simba\SimbaEngineSDK\8.0\Databases\Text

Figure 9: The DSNs installed with SimbaEngine SDK or available in .INI files.

You must compile the Example drivers before you can use them. However, they contain all the source code and project files or make files to create complete drivers that will work immediately. The purpose of these examples is to provide complete solutions so you can see how your custom solution will look when you are done. The SimbaEngine Codebase example driver is where you can find implemented examples of such SimbaEngine SDK features as indexes, bookmarks and Collaborative Query Execution.

Can I build a debuggable driver?

The libraries for SimbaEngine SDK components are available in debug and release versions. The debug versions of the libraries have code optimization turned off and debugging information included in the linkable objects. The release versions have code optimizations turned on and no debugging information included. You can build your code with the same compiler options to create either debug or release executables. Typically, you will build the debug versions until you are ready to test performance or release candidates. Very occasionally, there are differences in the operation of the debug and release compiled versions of code. This mostly shows up in memory operations because compiled debug code is typically more tolerant of memory problems, such as buffer overruns, than is release code.

3.2.1 SDK Examples

Here are short descriptions of the Example executables included in SimbaEngine SDK:

SimbaEngine Codebase Driver

The SimbaEngine Codebase driver is an ODBC driver DLL or shared object that includes Simba SQL Engine. Since the Codebase data store does not include SQL processing, Simba SQL Engine is required. The Codebase data store is ISAM-like and supports indexes and several data types. The files types are compatible with dBase files. An ODBC configuration DLL is included.

SimbaEngine Codebase Server

The SimbaEngine Codebase server is a stand-alone server that implements the same functionality as the SimbaEngine Codebase driver. The difference is that the build system links Simba SQL Engine against SimbaServer instead of SimbaODBC. The server accepts connection requests from SimbaClients and supports the Simba Client/Server protocol. You can use this build for remote testing of your Simba SQL Engine DSI implementation after you have it working as an ODBC DLL or shared library.

SimbaEngine Quickstart Driver

The SimbaEngine Quickstart driver is an ODBC driver DLL or shared object that includes Simba SQL Engine. Instead of reading Codebase files, the SimbaEngine Quickstart driver reads text files in tabbed Unicode text format. The intent of this driver is to provide a simple, working driver that you can transform into a driver that accesses your non-SQL data store. An ODBC configuration DLL is included.

SimbaEngine Quickstart Server

The SimbaEngine Quickstart server is a stand-alone server that implements the same functionality as the SimbaEngine Quickstart driver. The difference is that the build system links Simba SQL Engine against SimbaServer instead of SimbaODBC. The server accepts connection requests from SimbaClients and supports the Simba Client/Server protocol. You can use this build for remote testing of your Simba SQL Engine DSI implementation after you have it working as an ODBC DLL or shared library.

3.2.2 SDK Pre-built Binaries

There are two additional examples included: two ODBC clients and one JDBC client that connect to SimbaServer. Included in their folders are their configuration DLLs, where appropriate. You can also find reference versions of the installation executables in the "Bin" folder so you always have a copy you can install. Here are short descriptions of these clients and their installers:

SimbaClient for ODBC

SimbaClient for ODBC is an ODBC driver DLL or shared object that can connect to SimbaServer. It includes SimbaODBC and a DSI implementation that communicates via the Simba Client/Server protocol with SimbaServer. Since any SQL engine in the stack will be on the server side, there is no need for Simba SQL Engine in this driver. This is a completely generic ODBC driver that, when queried, reports the capabilities of the database that is connected to SimbaServer. There is nothing to modify in this ODBC driver.

SimbaClient for JDBC

SimbaClient for JDBC is a JDBC 3.0 driver packaged as a Jar file so you can install it into your Java Run Time Environment. Instructions and supporting files are included so you can help your customers install this driver. SimbaClient for JDBC includes the equivalent of SimbaODBC and custom Java code that communicates via the Simba Client/Server protocol with SimbaServer. It has the same relation to SimbaServer as SimbaClient for ODBC. There is nothing to modify in this JDBC driver.

SimbaD2O

SimbaD2O is a stand-alone server that provides a bridge to any ODBC enabled driver, including drivers based on the SimbaEngine 7 SDK. The server accepts connection requests from SimbaClients and supports the Simba Client/Server protocol. You can use this build for remote testing of any of your non-server DSI implementation drivers or any other ODBC driver you may have. For more information on how to setup and configure the SimbaD2O driver, please refer to the ReadMe provided.

3.2.3 Using the Examples

To test any of these examples by building and running them, the procedure is approximately the same:

1. Navigate to the appropriate folder
2. Open the solution file in Visual Studio or edit the make file
3. Verify that the configuration is what you want – debug/release, statically/dynamically linked system files, 32-bit/64-bit (note that these come in different versions of the SDK)
4. Build the executable

For stand-alone ODBC drivers:

5. Use the installed DSN to connect to the driver
6. Verify that you can see tables by retrieving metadata (SQLTables, SQLColumns)

7. Verify that you can see the data by executing a simple query like `SELECT * FROM T1` (SQLExecDirect)

For SimbaServer:

1. Install one of the SimbaClients – ODBC can be the most convenient
2. Start the server process
3. Create a DSN on the client machine (can be the same machine as the server machine)
4. Confirm that the client can connect to the server
5. Verify that you can see tables by retrieving metadata (SQLTables, SQLColumns)
6. Verify that you can see the data by executing a simple query like `SELECT * FROM T1` (SQLExecDirect)

To build your own driver using the SimbaEngine Quickstart driver as a starting point, begin by making a copy of the appropriate folder and renaming it. Then follow the “Build a Driver in Five Days” instructions, modifying your newly created SimbaEngine Quickstart sample. You can continue to add and test new functionality to your SimbaEngine Quickstart sample. The most commonly used functions are the use of indexes, insert and update, delete, create and drop tables, and create and drop indexes. To build a commercial driver you might want to pursue a more formal path with analysis and design before doing too much implementation because the early decisions you make can effect later decisions and implementation.

Unit testing, developer testing

One of the most effective tools for testing an ODBC driver is ODBCTest, a GUI application that allows the developer to make any ODBC call, with any parameter value, in any order. The results of each call are displayed – errors or retrieved data and metadata. The original is available in the Microsoft Data Access Components download on the Microsoft web site. Search for “MDAC” on MSDN and follow the links. There is also a version that works on Linux available from The UnixODBC Project. Look on www.unixodbc.org.

3.2.4 Troubleshooting

A common cause of failure to connect to the data store is an ODBC driver or a server process that cannot find the ICU DLL or shared object and refuses to start. This can be frustrating to diagnose so watch out for it. It is the general case of the driver not being able to find all its dependencies. On Windows, this manifests itself as a -1 Error. One way to approach this problem is with Dependency Walker. This free program identifies the items on which an executable depends. There is an article in MSDN about it here <http://msdn.microsoft.com/en-us/magazine/bb985842.aspx> and you can find the application here <http://dependencywalker.com/> .

Another cause of failure is the server or ODBC driver being unable to find your data store. This is usually a case of configuring the driver or server wrong. Make sure to include the right checks in your DSI implementation code to detect this condition, and to return clear error messages to the user. This is a frustrating problem to diagnose because the cause is often buried at the very bottom of the data access stack.

3.2.5 Call Us If You Need Help

However you proceed, please remember that the Simba Support team wants you to succeed and are delighted to answer any question you may have. Look in Appendix D: How Do I Get Support? for the information to contact us.

3.3 How does each component work?

This section provides a brief explanation of how the main SimbaEngine SDK components work. This is to help you understand how the SDK works and how the components work together. For more detail on how a component works please read the documentation on that component.

3.3.1 SimbaODBC

SimbaODBC provides a complete ODBC 3.52 interface and all the processing required to meet the ODBC 3.52 specification. We designed it as the connection between your custom DSI implementation and common reporting applications such as Crystal Reports and MS Excel. All that is required to create a complete ODBC driver is a DSI implementation that connects to an SQL database. SimbaClient for ODBC is an example of a driver like this. You can create a more complicated driver for non-SQL data by adding Simba SQL Engine to the driver. The SimbaEngine Codebase Driver is an example of this kind of driver.

To learn more about ODBC and building drivers for SQL-enabled data stores, please read the **SimbaODBC User's Guide**. For detailed information about the DSI API method calls, please see the on-line documentation found in the Documentation folder in the installed SimbaEngine SDK.

3.3.2 SQL Engine

Simba SQL Engine is a self-contained SQL parser and execution engine. It consumes ODBC SQL-92 queries, parses them, creates an optimized execution plan, allows the DSI implementation to take over part or all of the execution, and then executes the plan against the DSI implementation. We have uniquely designed Simba SQL Engine to be sandwiched between two instances of the DSI rather than exposing another API. The top end of SQL Engine is compatible with the SimbaODBC or SimbaServer DSI specification and you can link it directly to either of them to create a stand-alone ODBC driver or a server. The bottom end of Simba SQL Engine is compatible with another part of the DSI specification. It calls into a DSI

implementation that connects to a non-SQL data store. The result is SQL processing added to a non-SQL data store and made available to common reporting tools through standard interfaces.

Figure 10, below, illustrates the architecture of Simba SQL Engine at a high-level. The Preparation component contains the SQL parser. It also validates the SQL by making sure that data store objects such as tables and columns exist. When the SQL statement is prepared it is handed to the Execution component. The Execution component provides the environment for executing the execution tree and retrieving the result set. To learn more about Simba SQL Engine and building drivers for non-SQL capable data stores, please read the SimbaEngine User's Guide. NOTE: THE SimbaEngine User's Guide IS NOT YET PUBLISHED.

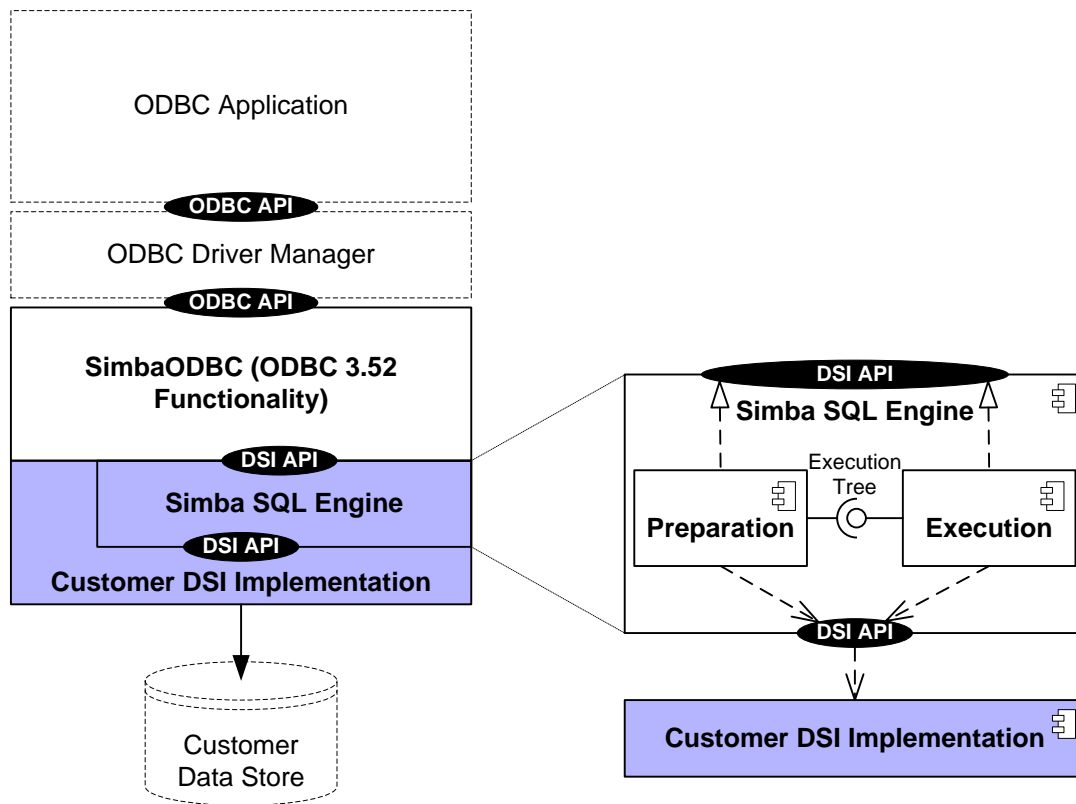


Figure 10: A look inside Simba SQL Engine showing the Preparation and Execution components communicating via the Execution Tree.

Simba SQL Engine expects to have its non-SQL data store presented to it as tables and columns. This is part of the relational, SQL view of data, and the job of the DSI implementation is to translate from the actual data storage schema to a table and column view. Put another way, if you can translate your data store into a view with tables and columns, then the addition of Simba SQL Engine will create a relational RDBMS. If the data store will support it, an RDBMS created with Simba SQL Engine will support DDL, DML, and DCL SQL statements.

How can I help Simba SQL Engine to optimize queries?

If the data store implements indexes of some kind, or if it is able to find specific data rows very quickly, then Simba SQL Engine can take advantage of this data store functionality to optimize operations such as filtering and sorting. Note that the data store does not actually have to implement indexes in the traditional, ISAM, sense. As long as the data store can seek to specific data rows *as if* it was using an index, that is, faster than the SQL Engine could step through the data itself, then Simba SQL Engine can use that functionality as if it was a real index. Simba SQL Engine also uses table cardinality and other metadata to optimize the query before execution. If indexes and table cardinality are not available, Simba SQL Engine will still work but it will be slower because it will not be able to perform the more advanced optimizations.

Simba SQL Engine uses the index and metadata information in a sophisticated optimizer to find the lowest-cost execution plan. However, many data stores have high performance features that are part of the value they deliver to users. With Simba SQL Engine's Collaborative Query Execution, you can use these features to accelerate the execution of SQL statements under Simba SQL Engine.

What kind of processing will Simba SQL Engine let me do in my data store?

Before it executes an SQL statement, Simba SQL Engine can pass to the DSI implementation an optimized representation of the SQL statement, called an Algebraic Expression Tree, or AE-Tree. The SQL statement takes this form just before Simba SQL Engine transforms it into an execution plan and executes it. When Simba SQL Engine passes the AE-Tree to the DSI implementation, the DSI implementation can choose to execute any part of the AE-Tree itself. It signals its intentions by modifying the AE-Tree before returning it to Simba SQL Engine. For instance, if the data store can filter data, or join data, or execute aggregate functions especially fast, it can modify those nodes of the AE-Tree to point to the DSI implementation for execution. The DSI implementation can modify any part of the AE-Tree if it can perform the execution quickly, or it can replace the entire tree and execute the whole query itself.

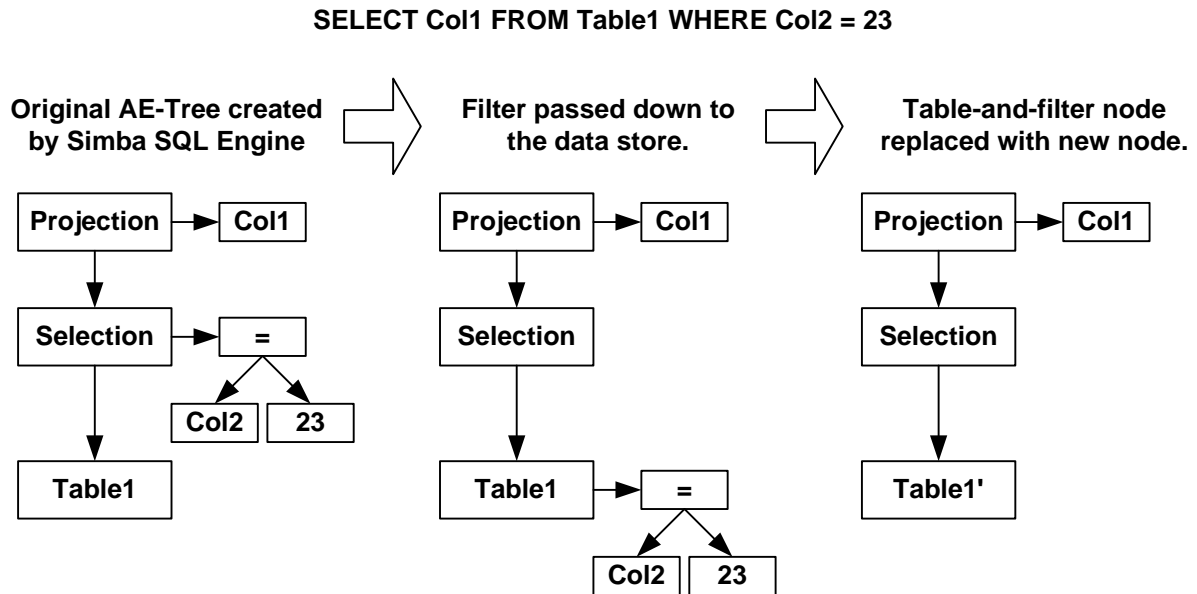


Figure 11: The progression of creating an AE-Tree, passing down a filter to the data store, and replacing the original table node with a new filtered table node.

This process is illustrated in Figure 11, above, which shows three views of a notional AE-Tree corresponding to the simple SQL query “SELECT Col1 FROM Table1 WHERE Col2 = 23”. The first view shows the AE-Tree originally created by Simba SQL Engine. In this case all the columns in the projection are retrieved and filtered by the Simba SQL execution engine. In the second view, the filter has been passed down to the DSI implementation code and the filter function removed from the selection node. Now the DSI implementation is responsible for filtering the rows it returns to Simba SQL Engine. The DSI implementation replaces the “Table1” node with a new node, “Table1’ ”, that only returns the filtered row set.

After the DSI implementation passes back the AE-Tree, Simba SQL Engine transforms the modified AE-Tree into an execution plan and executes it. Simba SQL Engine execution engine, and the DSI implementation and data store collaborate on the execution of the SQL statement, with the data store executing the parts it can do quickly, and Simba SQL Engine executing the rest. Of course, the DSI implementation does not have to modify the AE-Tree at all. Simba SQL Engine can execute the entire SQL statement quickly and efficiently by itself.

3.3.3 Client/Server

Simba Client/Server is a collection of smaller components that between them allow remote access to your data store. SimbaServer is most frequently used as a stand-alone executable, although it can be set up as a DLL or shared object under another server. You must link SimbaServer to a DSI implementation before it can be an executable, or you can use SimbaD20. The DSI implementation can include Simba SQL Engine or not, and it can be written to perform a wide range of functionality including SQL query processing with Simba SQL Engine, concentrating client requests through one executable, aggregating data stores, or

controlling data access through role-based permissions. There are many possibilities for using SimbaServer as an intermediate processing step in a larger system.

Running SimbaClient and SimbaServer

When you start up your linked SimbaServer, it binds to a configurable port on your server machine and listens for connection requests from SimbaClient drivers. Since all SimbaClient drivers use the same protocol they are all handled in the same way by SimbaServer. When a SimbaClient driver finds SimbaServer, it requests a TCP/IP connection. With a successful network connection, the SimbaClient and SimbaServer begin a conversation using the Simba Client/Server protocol. This is a layered protocol designed for clients making remote data queries and optimized for transmitting the result sets back to the client. It is independent of the standard interface used by the user application. Figure 12, below, shows the relationship of SimbaClient and SimbaServer using SimbaClient for ODBC as an example.

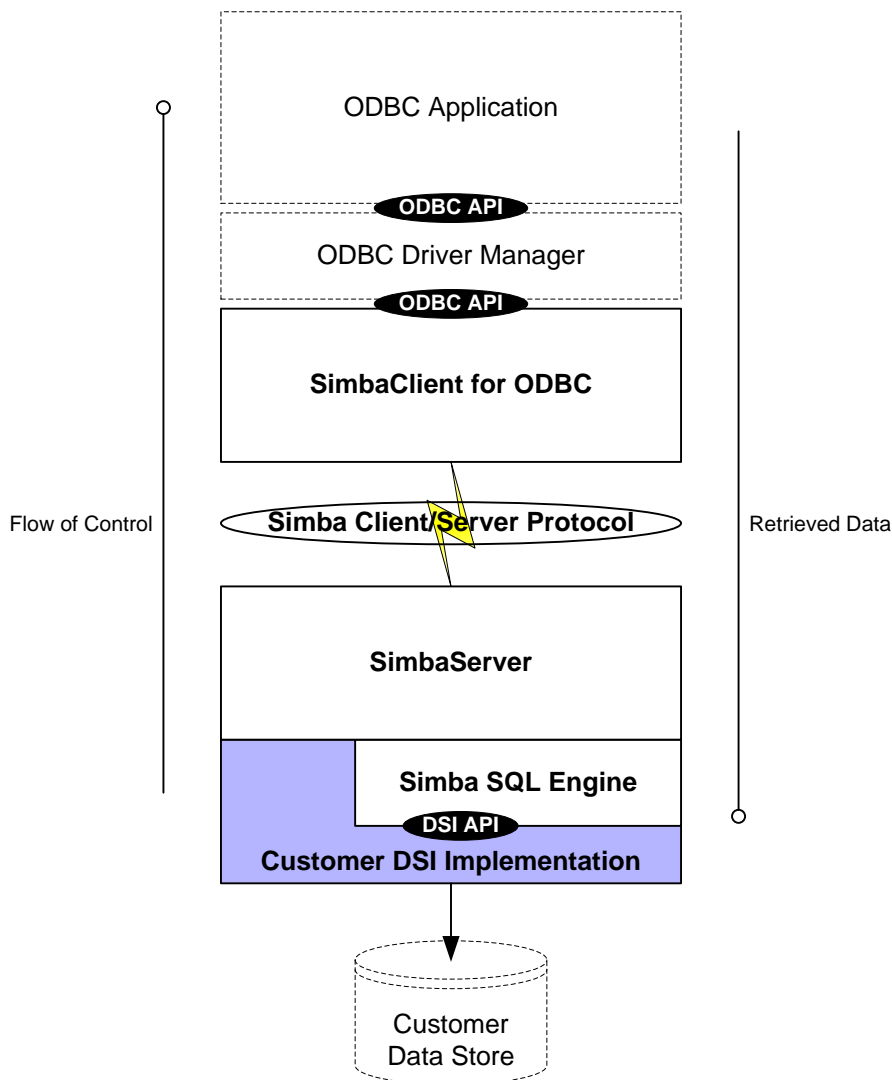


Figure 12: The architecture of Simba Client/Server showing the Client/Server protocol connecting the components.

SimbaServer is designed to optimize use of shared server resources, while SimbaClient is designed to optimize the responsiveness of the application to give the best experience to the user. The protocol parameters are configurable in case the default parameters do not provide the best performance for your circumstances.

Reusing your DSI implementation

One of the important design features of SimbaServer is that it links to exactly the same DSI implementation as SimbaODBC. This means that you can develop and test your DSI implementation as a stand-alone ODBC driver using SimbaODBC. This is a much simpler environment than developing using SimbaServer. When your new DSI implementation is performing to your satisfaction, you can link it to SimbaServer and begin testing it in a client/server environment. If you know the state of the logic and performance of the DSI implementation before introducing client/server, you can reduce your investigation time and debugging costs.

SimbaEngine SDK contains a SimbaClient for ODBC and a SimbaClient for JDBC that provide direct access to SimbaServer. ADO.NET applications can connect to SimbaServer via the Microsoft ADO.NET/ODBC bridge.

3.3.4 SimbaEngine Codebase Driver Sample

The SimbaEngine Codebase example is a fully worked example of a DSI implementation connecting to an ISAM-like data store. In this case, the data store is the Codebase database that uses dBase format files and indexes to store data. This is exactly the way that the DSI looks at data stores so there is little translation here. However, one of the purposes of the SimbaEngine Codebase DSI implementation is to provide examples of features you can examine, copy and adapt to your own situation. Much of this functionality has to be in all DSI implementations so it is useful to see it in action. The kinds of functionality illustrated in the SimbaEngine Codebase sample DSI implementation are:

- Driver-wide information
- Opening tables
- Retrieving data from tables
- Virtual tables
- Metadata – gathering it and retrieving it
- Indexes
- Table cardinality (for optimization)
- Data caching in the DSI implementation
- Collaborative Query Execution optimization of filters, joins and aggregation

Verifying suspected defects in SimbaEngine SDK

Another purpose of the SimbaEngine Codebase sample DSI implementation is to provide a canonical DSI implementation that can be used to analyze and debug problems encountered. For instance, if you think your DSI implementation is working correctly but there is a problem in your SimbaEngine SDK system, there is a simple way to determine where the problem lies. If the problem shows up when you run the system you have assembled with the SimbaEngine Codebase sample DSI implementation, then the problem lies in SimbaEngine SDK components. In addition, we can easily duplicate and fix the problem. If the problem goes away when you remove your DSI implementation from the system then you need to do some more investigation. In either case, analysis and debugging is focused and reduced, lowering your cost to deliver a solution to your customers.

3.3.5 SimbaEngine Quickstart Driver Sample

The SimbaEngine Quickstart example is a minimal DSI implementation that nonetheless works and retrieves data from a data store. It is simple enough for you to understand its operation quickly and to modify it step-by-step to begin to retrieve data from your data store. We designed the SimbaEngine Quickstart driver primarily for prototyping DSI implementations so you can quickly understand the way SimbaEngine SDK works and get hands-on experience. However, 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 the fastest way to get a data access solution to your customers.

SimbaEngine Quickstart data files

The SimbaEngine Quickstart example uses tabbed Unicode text files that can be created by MS Excel as its data store. With Simba SQL Engine, it is fully capable of retrieving, filtering and joining data from these tables, but the tables limit the data types supported and the performance. The functionality of the SimbaEngine Quickstart example allows you to make incremental changes to the code and then test to make sure that everything still works. This is a proven way to understand the working of the DSI implementation and to prototype a DSI implementation for your own data store. This is also the best way to explore what you need to translate the schema of your data store to the tables and columns representation required by the DSI interface.

There is a well-developed series of steps to take to get a prototype DSI implementation working with your data store; setting up the development environment, making a connection to the data store, retrieving metadata, working with columns, retrieving data, and writing data. At the end of each step, you can verify that the code you have written is working by testing the driver against your data store. At the end of five days, you will have a read-only driver connecting to your data store. This will not be a commercial driver, but you will have learned a lot about SimbaEngine SDK, and if you want to you can move on to add write capability and optimizations to create a first version of a commercial driver. Read Section 4, Build an ODBC Driver in Five Days, to follow this methodology.

4 Build an ODBC Driver in Five Days

This section describes the steps required to build a prototype ODBC driver on Windows that uses Simba SQL Engine and your data store. You should allow up to five days to complete all the steps described here. Although you may finish much sooner, this is typically enough time to have a working prototype driver ready to demonstrate.

The steps involve changing the Data Store Interface (DSI) implementation of the SimbaEngine Quickstart Driver so that it accesses your data store instead of the example data store. We have organized the steps into five separate days of work. If you can finish any day's work in less than one day, you can move ahead to the next day's work.

In the SimbaEngine Quickstart Driver code we have highlighted the areas you need to change by adding #pragmas that include the keyword "TODO" so you can find them, along with a short explanatory message. When you compile the code these #pragmas show up in the output window. There are 10 areas in the code highlighted with TODOs. You will visit each one.

Of the 10 areas of the code that you need to modify, eight changes are for productization rather than actually connecting your data store to Simba SQL Engine. These are things like naming the driver, setting the properties that configure the driver, and naming the XML error file and log files. In other words, these are not complicated tasks.

The other two areas of the code you will modify actually handle the data and metadata. They are mostly concerned with getting the data and metadata from your data store reader and into the Simba SQL Engine. You are the acknowledged expert on your data store, so what remains is getting the data and metadata into the Simba SQL Engine. Since the Simba Quickstart Driver already has the classes and code to do this, all you have to do is modify what already exists and your driver will begin to work.

The simplicity of the DSI API also helps you because there is order and symmetry to the way it works. The UML diagram below shows the design pattern to look for. Almost all DSI implementations wind up with a similar pattern. Look for the circular pattern of class relationships headed by IResult and anchored by your Utilities classes. These are called QUtilities in the Simba Quickstart Driver. If your DSI implementation design looks like this, you are on the right track.

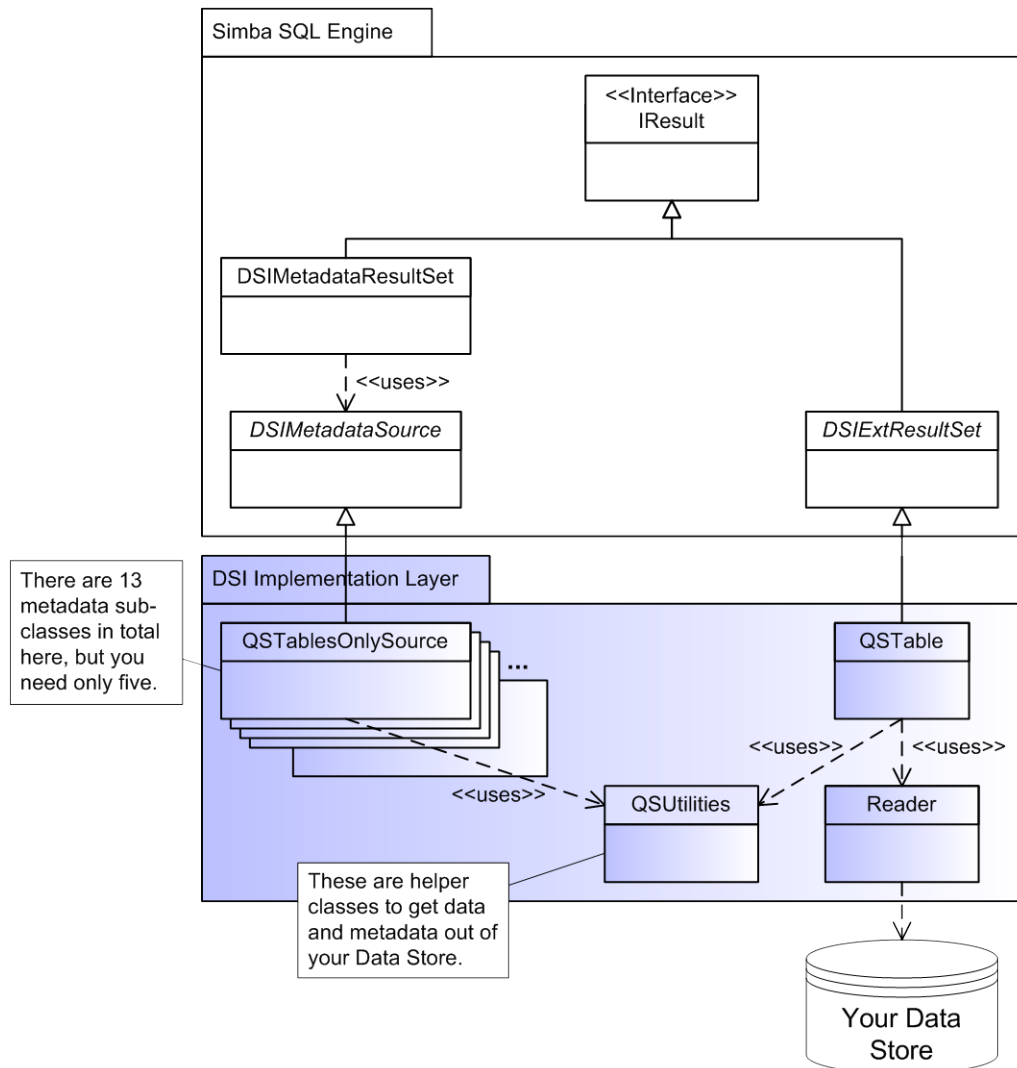


Figure 13: A high-level view of the generally-successful design pattern for a DSI implementation driver.

Implementing data retrieval is straightforward. Your Reader class interacts directly with your data store to retrieve the data and deliver it to the QSTable class on demand. The Reader class should take care of caching, buffering, paging, and all the other techniques that speed data access. 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 only implement the type information MetadataSource in addition to the DSIExtMetadataHelper and return NULL 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 5:00 PM Pacific Time, Monday to Friday. Dial 604-633-0008 and select 3. 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 oriented 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)

4.1 Day One

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, 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.
2. Take a few minutes to read this 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 - one for the release version of each of the two example ODBC drivers. These are the SimbaEngine Codebase Driver and the 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 MS Access, MS Excel or ODBCTest32.exe (Unicode).

Build and Test the Example Drivers

1. Before you begin to compile and debug the examples, you should edit the appropriate registry (.reg) file located in your development folder. When you run this file, it will update your Windows registry to register the combined ODBC driver and configuration DLL you are about to create. There are three registry files: one for 32-bit Windows, one for a 32-bit ODBC driver on 64-bit Windows, and one for a 64-bit ODBC driver on 64-bit Windows. Which registry file you use depends on the environment in which you want to use your new driver. For more information about 32-bit versus 64-bit drivers, please see Appendix C: Windows 32-Bit vs. 64-Bit.

When you have successfully created the driver and configuration DLL, you will be able to use the ODBC Data Source Administrator, which is part of the Windows operating system, to create and modify DSNs that use your newly created driver. Note that we have combined the DSN configuration functionality and the ODBC driver into one DLL for convenience. You can separate these functions into different DLLs if you want.

2. The source for the example ODBC drivers is located in the Examples folder. These instructions assume you will be working with the 32-bit version of the Visual Studio 2005 SimbaEngine Quickstart Driver example on a 32-bit Windows operating system.

3. From the Main Menu select Build->Configuration Manager and make sure that the Active solution configuration is Debug. Now select Build->Build Solution (F7) to build the driver. This will build the debug version of the driver.
4. Run the ODBC Data Source Administrator. To do this, click on Start -> Settings -> Control Panel. Select Administrative Tools, and then select Data Sources (ODBC). In the System DSN tab When the Data Source Administrator opens, click on Add. This will open the Create New Data Source dialog. Select MyQuickstartDSIIDriver and click on Finish. This will bring up the Quickstart Data Source Configuration dialog from the driver you just created. Give your new DSN a name. You can give it a description as well if you want. Then click on Browse and navigate to C:\Program Files\Simba\SimbaEngineSDK\8.0\Databases\Text . This is where the comma-separated-values (.csv) database files are that the Quickstart sample driver can read. Click on OK to create your new ODBC DSN.
5. Open any ODBC-enabled application (e.g.: ODBC Test) and, with your Quickstart driver solution open, 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 Simba Engine 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 SimbaEngineQuickstart directory and paste it to the same location. This will create a new directory called "SimbaEngineQuickstart - 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.
2. Open your new directory, and rename the .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 (.vcproj) and replace every instance of "QuickstartDSII" in the source code with the name of your new ODBC driver. Then save and close the file. Do the same for the solution file (.sln) as well, making sure that the project file name used is correct.
4. You will need to register your new ODBC driver and create a new ODBC DSN before you will be able to test it. Edit a copy of the appropriate registry (.reg) file located in your development folder and merge it with the Registry. This will register the driver and then, using the ODBC Data Source Administrator, add a new ODBC DSN that uses it. Point to the new ODBC driver file you will create for registration and for the new ODBC DSN. For

now, also point the new ODBC DSN to the Text database folder so the Quickstart code will continue to work until you modify it.

5. Build the project to make sure everything compiles. At this point, the new DSII project is identical to the SimbaEngine Quickstart Driver example.
6. 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 component ID and package 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.

4.2 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::SetDriverPropertyValues()` 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 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`. You can also use the utility functions supplied: `VerifyRequiredSetting()` and `VerifyOptionalSetting()`. 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`. You can also use the utility functions supplied: `GetRequiredSetting()` and `GetOptionalSetting()`.

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

4.3 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 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: *GetData()* (See section 4.6, Technical Note: Data Retrieval, below 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:

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

3. You may need to change the types returned and the parameters for the types in QSTypeInfoMetadataSource::InitializeData().

Handling the other MetadataSources

1. When called with any other DSIMetadataTableId, QSDataEngine::MakeNewMetadataTable() should return NULL. Returning NULL will signal SimbaEngine SDK that it should use the metadata helper class returned via QSDataEngine::CreateMetadataHelper() along with some default MetadataSources to create the data source metadata.
2. You will need to change:
 - a. QSMetadataHelper::QSMetadataHelper()
The example constructor does retrieves a list of the tables in the data source. You should modify this method to load the tables defined within your data store.
 - b. QSMetadataHelper::GetNextTable()
In the SimbaEngine Quickstart Driver, this method returns the next table in the data source. You should modify this method to retrieve the next table from your data store.
 - c. The DSIMetadataHelper class works by retrieving the identifying information for each table and then opening the table via QSDataEngine::OpenTable(). Once you have implemented QSTable, the correct metadata will be returned for all of the tables and columns in your data source.

Congratulations! You can now retrieve type metadata from within your data store. Once you have completed the work for Day Four, you will be able to retrieve the full set of metadata from your data store. You should be able to run SQLGetTypeInfo() from within ODBCTest32.exe (Unicode) and see the correct metadata returned.

4.4 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.

QSDataEngine::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 DSIMetadataHelper, 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:

```

AutoPtr<DSIResultSetColumns> columns;
  Get all the column information from your data store for the table
  For Each Defined Column
  {
  AutoPtr<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 which provides an interface to navigate between lines within a Unicode text file. This class preprocesses each row in the file to determine the starting file offset of each column in the row. Its GetData method takes a 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.

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

4.5 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 component ID and package name.
```

All error messages returned by the driver are stored in the same package within the messages XML file. The reference to the package within your DSI implementation is identified by “QS_ERROR”. Simply change the “QS” to your chosen two-letter abbreviation.

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. 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.

4.6 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's, 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.

4.7 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. `QSTable::SetConnectionPropertyValues()` currently disables schema support via `DSIPropertyUtilities::SetSchemaSupport()`. Change this value to `true` to enable schema support.

2. You will also need to change:
 - a. `QSMetadataHelper::GetNextTable()`

In the SimbaEngine Quickstart Driver a blank schema is returned as schema support is not enabled by default. The schema will need to be returned in the Identifier to allow SimbaEngine SDK to open the correct table.
 - b. `QSDataEngine::OpenTable()`

Modify this method to verify the given schema and return the correct table for the given catalog, schema, and table name.
 - c. `QSTable::GetSchemaName()`

Modify this method to return the schema the table belongs to.

5 Frequently Asked Questions

This section answers the most commonly asked questions about our product and technology. These questions are divided into distinct categories. Simply click on the question from the category of your choice.

ODBC

- [What is ODBC?](#)
- [What is MDAC?](#)
- [I am new to ODBC. How does my application work with an ODBC Driver?](#)
- [What is ICU?](#)

Product

- [What is SimbaODBC?](#)
- [What do the different components of SimbaODBC do?](#)
- [How do I build an ODBC driver using SimbaEngine?](#)
- [How can I obtain more information about SimbaEngine?](#)
- [What SQL conformance level does SimbaEngine SDK support?](#)

What is ODBC?

ODBC stands for Open Database Connectivity (ODBC). It is a C-language open standard Application Programming Interface (API) for accessing relational databases.

In 1992, Microsoft contracted Simba to build the world's first ODBC driver; SIMBA.DLL, and standards-based data access was born. Using ODBC, you can access data stored in many common databases. A separate ODBC driver is needed for each database to be accessed. An ODBC Driver Manager is also needed. This is supplied with the Windows operating system, and is available commercially and as open source on UNIX and Linux.

What is MDAC?

MDAC stands for Microsoft Data Access Components. It is shipped with the Windows operating system and contains interfaces for ODBC, OLEDB and ADO and the ODBC drivers for Microsoft's database related products.

I am new to ODBC. How does my application work with an ODBC Driver?

ODBC-enabled applications always access ODBC Drivers through the Driver Manager that is installed on the operating system. An instance of the Driver Manager is created for each ODBC application. The application will specify to the Driver Manager which ODBC Driver to use when establishing a connection. The Driver Manager will then load the appropriate ODBC Driver. Once the ODBC Driver is loaded, the Driver Manager will map all incoming requests to the appropriate functions exported by the ODBC Driver.

To interact with a Driver Manager, ODBC-enabled applications will request the following three ODBC handles:

- `SQL_HANDLE_ENV`

Represents an environment handle. Every instance of an ODBC driver will be associated with a single environment handle.

- `SQL_HANDLE_DBC`

Represents a connection handle. Connections are created using one of the following three ODBC methods: `SQLConnect()`, `SQLBrowseConnect()`, `SQLDriverConnect()`. Every connection handle is associated with its parent environment handle.

- `SQL_HANDLE_STMT`

Represents a statement handle. Every statement that is to be executed via ODBC will be associated with its own statement handle. Every statement handle is associated with its parent connection handle.

The Driver Manager interacts with an ODBC Driver in much the same way. The Driver Manager will request the handles for the environment, connection and statement. All calls made from the ODBC-enabled application to the Driver Manager require the Driver Manager allocated handle and will be implemented as follows:

- Map incoming Driver Manager allocated handle to an instance representing the handle.
- Call the ODBC Driver associated with the instance using the ODBC Driver associated handle.

What is ICU?

ICU stands for The International Components for Unicode (ICU) libraries. These libraries provide Unicode handling mechanisms on which the SimbaODBC components are dependent. These libraries are distributed under an open source license at:

<http://source.icu-project.org/repos/icu/icu/trunk/license.html>

ICU is freely available from:

<http://www.icu-project.org/download>

What is SimbaODBC?

SimbaODBC is a component part of the SimbaEngine SDK for developing full-featured, optimized ODBC 3.52 drivers on top of any SQL-enabled data source. SimbaODBC provides extensibility for JDBC, OLE DB as well as ADO.NET connectivity. The SimbaEngine User's Guide will provide you with sufficient information to rapidly develop a read-only driver, which can then be extended to include additional functionality and optimizations. SimbaODBC simplifies exposing the query parsing, query execution and data retrieval facilities of your SQL-enabled data source.

What do the different components of SimbaODBC do?

SimbaODBC ships with a number of static libraries. You will link these libraries into the code you write to communicate with an underlying SQL-92 enabled data store.

How do I build an ODBC driver using SimbaEngine?

You can build an ODBC Driver for a SQL-enabled data store using the SimbaODBC component by referring to the SimbaEngine User's Guide. To build an ODBC Driver for a non-SQL-enabled data store, please refer to How to Build a Driver in 5 Days.

How can I obtain more information about SimbaEngine SDK?

Please visit our website at <http://www.simba.com> for additional resources, or contact us if you have any specific questions.

What SQL conformance level does SimbaEngine SDK support?

The SimbaEngine SDK supports the full core-level ODBC 3.52. It supports most of the Level 1 and Level 2 API.

Conformance Level	INTERFACES	Conformance Level	INTERFACES
Core	SQLAllocHandle	Core	SQLGetInfo
Core	SQLBindCol	Core	SQLGetStmtAttr
Core	SQLBindParameter	Core	SQLGetTypeInfo
Core	SQLCancel	Core	SQLNativeSql
Core	SQLCloseCursor	Core	SQLNumParams
Core	SQLColAttribute	Core	SQLNumResultCols
Core	SQLColumns	Core	SQLParamData
Core	SQLConnect	Core	SQLPrepare
Core	SQLCopyDesc	Core	SQLPutData
Core	SQLDescribeCol	Core	SQLRowCount
Core	SQLDisconnect	Core	SQLSetConnectAttr
Core	SQLDriverconnect	Core	SQLSetCursorName
Core	SQLEndTran	Core	SQLSetDescField
Core	SQLExecDirect	Core	SQLSetDescRec
Core	SQLExecute	Core	SQLSetEnvAttr
Core	SQLFetch	Core	SQLSetStmtAttr
Core	SQLFetchScroll	Core	SQLSpecialColumns
Core	SQLFreeHandle	Core	SQLStatistics
Core	SQLFreeStmt	Core	SQLTables
Core	SQLGetConnectAttr	Level 1	SQLBrowseConnect
Core	SQLGetCursorName	Level 1	SQLPrimaryKeys
Core	SQLGetData	Level 1	SQLProcedureColumns
Core	SQLGetDescField	Level 1	SQLProcedures
Core	SQLGetDescRec	Level 2	SQLProcedureColumns
Core	SQLGetDiagField	Level 2	SQLColumnPrivileges
Core	SQLGetDiagRec	Level 2	SQLDescribeParam
Core	SQLGetEnvAttr	Level 2	SQLForeignKeys
Core	SQLGetFunctions	Level 2	SQLTablePrivileges

ODBC 3.52 Interfaces Supported by SimbaODBC.

The ODBC version 3.52 specification provides three levels of SQL grammar conformance: Minimum, Core and Extended. Each higher level provides more fully implemented data definition and data manipulation language support. The level of supported SQL grammar is dependent on your SQL-enabled data source. At the very least, your SQL-enabled data source must conform to the minimum SQL grammar defined by the ODBC version 3.52 specification.

Appendix A: Platforms and System Requirements

The following are the minimum requirements for installing and working with SimbaEngine SDK.

- 10 GB of free disk space
- 1 GB RAM

On Windows Platforms

- Microsoft Visual Studio 2005 or 2008
- Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7.
- On Linux and UNIX Platforms C++ compiler (e.g. gcc, Sun Studio)
- Red Hat, SuSE, Solaris, AIX, HP-UX

Appendix B: Installation

SimbaEngine is delivered to you as a single self-extracting installation program (on Windows) or as a single tar.gz file (on UNIX). Please refer to the corresponding section depending on your platform.

On Windows Platforms

1. In Windows Explorer, double-click the installer executable file to run the installer. The version of the SDK is encoded in the name of the installer file.
2. Follow the instructions in the SimbaEngine installer.
 - a. Read and understand the license agreement.
 - b. Select the SimbaEngine installation location. The installer will install SimbaEngine and its components in a SimbaSDK directory at that location.
3. Click Finish.

On Linux and UNIX Platforms

On a UNIX platform SimbaEngine is provided as a single file consisting of the Simba*.tar.gz file, a tar format archive that has been compressed using the gzip tool. Note that the "*" represents a string of alphanumeric characters that represent the build number and platform of the kit. For the sake of brevity, we will refer to the kit as Simba.tar.gz.

1. Open a command prompt and change to a directory where you would like to install SimbaEngine.
2. To uncompress Simba.tar.gz, enter:
`gzip -d Simba.tar.gz` This will extract the file Simba.tar
3. To install SimbaEngine SDK enter:
`tar -xvf Simba.tar`

This will extract the contents of the file to a directory that is referenced to as <InstallDir> throughout the document.

Appendix C: Windows 32-Bit vs. 64-Bit

The SimbaEngine SDK installer is a 32-bit application so it can only create 32-bit data sources whether it is on a 32-bit or a 64-bit Windows machine. The sections below explain where the installer creates 32-bit data sources in various environments, and where to create 64-bit data sources.

32-Bit Platforms

32-bit Windows machines present a simple situation because they only support 32-bit drivers and applications. Installing an ODBC driver on a 32-bit Windows machine will result in new registry entries in HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBC.INI and HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI.

When the SimbaEngine SDK installer has installed the drivers and created new data sources, you can create more data sources and modify existing ones with the ODBC Data Source Administrator. To do this, click on Start -> Settings -> Control Panel. Select Administrative Tools, and then select Data Sources (ODBC).

64-Bit Platforms

64-bit Windows platforms are more complicated because you can install and use both 32-bit and 64-bit drivers. 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 sources created in different areas of the registry:

32-Bit Drivers

64-bit Windows can host both 64-bit and 32-bit applications and drivers. The 32-bit applications and drivers see a subsection of the registry that is separate from the 64-bit applications and drivers. The registry entries for 32-bit drivers and data sources are located under HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBC.INI and HKEY_LOCAL_MACHINE/SOFTWARE/WOW6432NODE/ODBC/ODBCINST.INI.

To create new 32-bit data sources or modify existing ones you can use the 32-bit ODBC Data Source Administrator. Run this by executing C:\WINDOWS\SysWOW64\odbcad32.exe. 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.

64-Bit Drivers

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 registry entries for 64-bit drivers on a 64-bit

machine will be located under HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBC.INI and HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBCINST.INI.

You can create more 64-bit data sources and modify existing ones with the ODBC Data Source Administrator. To do this, click on Start -> Settings -> Control Panel. Select Administrative Tools, and then select Data Sources (ODBC).

Applications

It is important to remember, when you are working on a 64-bit Windows machine 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. This can lead to confusing problems where a perfectly configured ODBC DSN does not work because it is loading the wrong kind of driver. In addition, the ODBC Data Source Administrator is available in 64-bit and 32-bit versions so you have to know what kind of ODBC data source you want to create.

A Word of Caution

Since you can execute 64-bit and 32-bit applications transparently on a 64-bit system, you might find that you are using a 32-bit application that you thought was 64-bit. In particular, Microsoft Office applications are 32-bit through 2009. If you are using anything but the latest version of Microsoft Excel it is likely to be a 32-bit application and it will not work with a 64-bit ODBC driver.

Appendix D: How Do I Get Support?

We welcome your questions and comments. To help us help you faster please have ready a detailed summary of your machine environment (operating system, version, patch-level, etc.) before you contact us. Providing us with this information helps us to understand your situation and to help you more effectively.

By telephone:

Customer Support: +1.604.633.0008 ext 3. Customer Support is available Monday to Friday, from 8 a.m. until 5 p.m. Pacific Time.

By fax or e-mail:

Fax: +1.604.633.0004

Send e-mail to support@simba.com

On the Web:

Visit us on the Web at www.simba.com and submit technical requests online at www.simba.com/support.htm