SimbaProvider SDK Technical White Paper

www.simba.com

Contents

Simba Technologies Inc.
938 West 8th Avenue
Vancouver, BC Canada
V5Z 1E5

Tel. +1 604 633 0008
Fax. +1 604 633 0004

solutions@simba.com

This white paper provides a technical overview of SimbaProvider SDK—the world's most compelling OLE DB for OLAP (ODBO) and XML for Analysis (XMLA) Provider development solution available.

Overview

SimbaProvider SDK contains a robust MDX 2005 engine and is the leading solution for OLE DB for OLAP (ODBO) and XML for Analysis (XMLA) connectivity to multi-dimensional OLAP or star-schema relational data sources. With SimbaProvider SDK, you can integrate your multi-dimensional server with the newest generation of analytic applications and development environments that use the above interfaces (e.g. SAP BusinessObjects, Cognos, Microsoft Excel and Reporting Services, ProClarity, ADOMD, ADOMD.NET, and many others). SimbaProvider SDK enables the development of robust 32-bit and 64-bit ODBO Providers and 32-bit and 64-bit .NET-based or Java-based XMLA Providers. This ensures the utmost in interoperability and crossplatform (Windows and Linux) enabled solutions—all you need to do is choose your development language and proceed to develop your data provider, as explained in SimbaProvider's comprehensive documentation.

Unlike any other product on the market, SimbaProvider SDK provides an open framework from which you can quickly and easily

Simba Technologies is a pioneer in OLE DB for OLAP (ODBO) and XML for Analysis (XMLA).

Over half of the ODBO Providers available today were built with Simba's technology.

develop quality ODBO and XMLA Providers. SimbaProvider SDK offers significantly reduced development time, simplified maintenance, and interoperability with leading applications that comply with the ODBO and XMLA specifications. SimbaProvider SDK's MDX engine provides your multi-dimensional data store with full MDX 2005 capabilities. All you have to do is follow our step-by-step development process to provide the world of open connectivity tools to your customers.

SimbaProvider SDK Highlights

Below are just a few highlights of SimbaProvider SDK:

Flexibility – Get both 32-bit and 64-bit OLE DB for OLAP (ODBO) and XML for Analysis (XMLA). For the same cost and development time you get both interfaces, including the ability to build both a .NET and Java-based XMLA Provider for the utmost interoperability on multiple platform environments.

Functionality – Start with a simple ODBO Provider or XMLA Provider and then add advanced functionality.

MDX 2005 Engine – Simba's MDX 2005 Engine can parse, resolve and evaluate multi-dimensional expression (MDX) queries providing optimal performance. SimbaProvider SDK is the only software development kit that supports the MDX 2005 query language.

Conformance – Simba takes care of your data provider's compatibility with the latest platforms, data access standards, and client reporting and analysis tools. This streamlined approach ensures that your ODBO Provider or XMLA Provider will work with the latest Business Intelligence (BI) tools and connect with all major multi-dimensional data sources.

Portability – SimbaProvider SDK's MDX 2005 Engine is developed in ANSI C++ and supports both Windows and Linux.

Step-by-Step Development Approach – Get your prototype ODBO Provider or XMLA Provider up and running quickly with our 10 step development process.

Comprehensive Development Tools – SimbaProvider SDK contains a sample ODBO Provider and XMLA Provider to accelerate development of your data provider. It also contains comprehensive test tools.

Comprehensive Documentation and Support – Receive a full suite of documents, including a developer's guide, technical reference materials, and online and phone support.

Ease of Installation – Installing SimbaProvider SDK is a simple process. Windows users will be led through an InstallShield installer. In minutes, the installation of the SimbaProvider SDK package is complete.

SimbaProvider SDK Benefits

SimbaProvider SDK provides solutions for the more difficult areas in implementing an ODBO Provider or XMLA Provider, making it easy for you to develop a robust data provider.

Dataset (Rowset) Flattening – Technically optional, dataset flattening is required in order to work with certain consumer applications. Data flattening requires joining axis info tables and dataset cell information. The flattened dataset provides a quick and easy method for rowset-based consumer applications, such as Microsoft Excel to display multi-dimensional data.

Asynchronous Execution and Rowset Population Support – Asynchronous execution and Rowset population requires the close coordination of several interfaces, (IRowset, ICommand, IAsynchStatus, and IConnectionPointContainer), as well as overall thread safety and thread scheduling/handling. This requires a good design and expert threading experience to implement. Without proper asynchronous execution support, applications reading the data may otherwise appear slow and ill-behaved to your end-users. SimbaProvider SDK supports asynchronous execution and Rowset population.

Full Fledged 32-bit and 64-bit MDX 2005 Engines – Simba's MDX 2005 Engines have the ability parse, resolve and evaluate MDX queries. Simba's MDX Engines support the latest MDX 2005 enhancements to ensure utmost interoperability, including compatibility with the newest applications. The MDX language is very difficult to implement. Specifically, it is much more complex than SQL. Parsing MDX is very difficult because of its many keywords and ambiguities. SimbaProvider SDK contains full MDX 2005 Engine code binary that is backed by Simba's commitment to keeping pace as technologies evolve.

Caching – Many applications access datasets one cell at a time. This is extremely inefficient and caching is highly desirable. Because datasets are multi-dimensional and can be quite large, dataset read-ahead and caching requires sophisticated algorithms to pre-calculate and retrieve those areas of the cube that are most likely to be retrieved next. SimbaProvider SDK implements caching and allows you to extend this caching if desired.

SimbaProvider SDK Architecture

SimbaProvider SDK allows you to quickly and easily build an Online Analytical Processing (OLAP) Provider. The SimbaProvider SDK includes a Multi-dimensional Expression (MDX) Engine that allows you to rapidly implement MDX 2005 support in your data provider.

SimbaProvider SDK contains a collection of libraries and sample source code to allow you to build your own ODBO or XMLA Provider. The source code that you write can be used to create either type of data provider, simply by building it with the appropriate Simba components.

SimbaProvider SDK is structured to provide optimum flexibility. Whether you choose to build an ODBO Provider or a XMLA Provider (.NET or Java), your choice need not affect the architecture of your implementation layers. This structure allows you to re-use the same implementation to create both types of data providers.

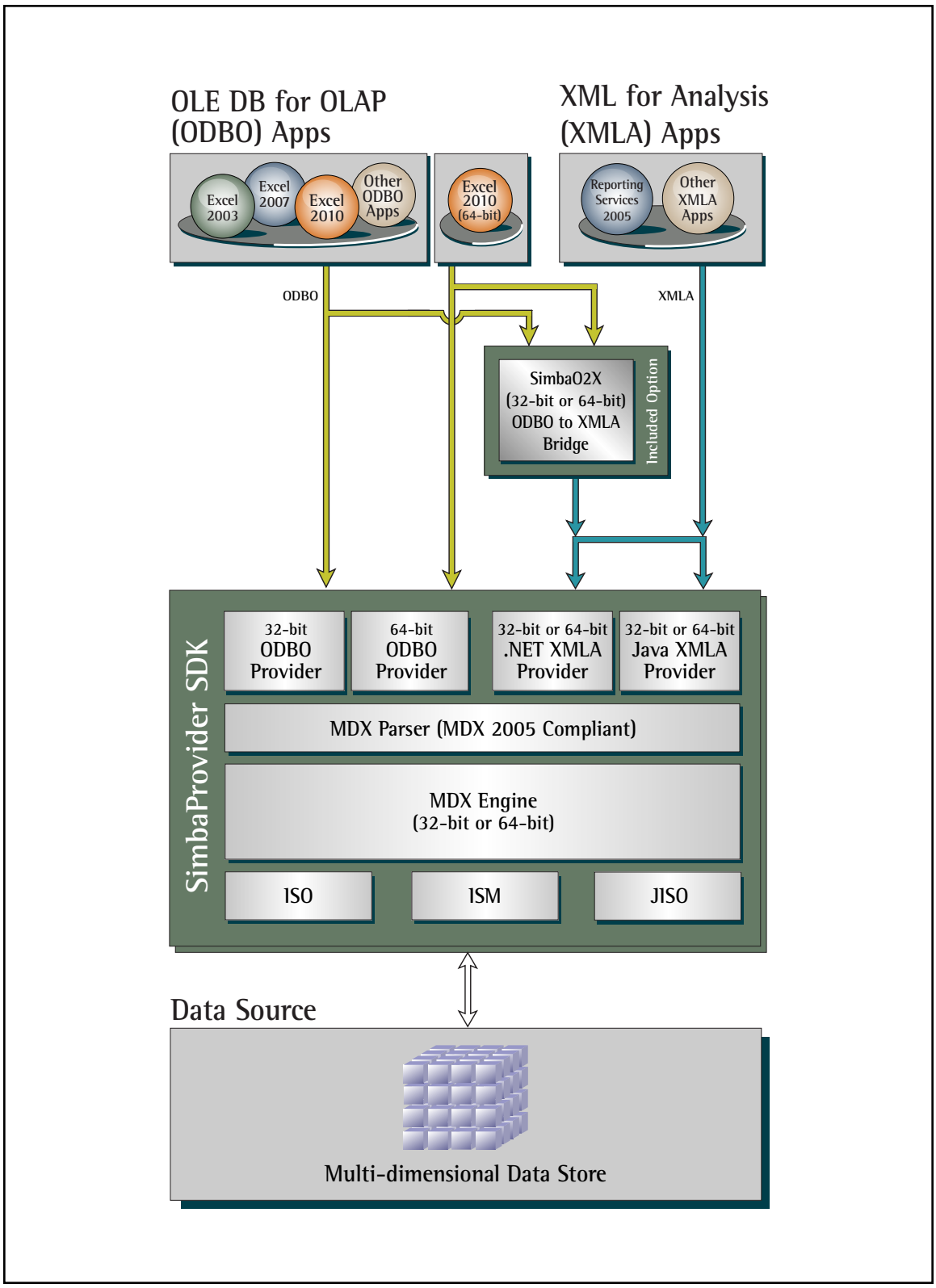

Figure 1. SimbaProvider SDK Architecture

OLE DB for OLAP (ODBO) Architecture

Figure 2 shows the conceptual relationship between an ODBO Consumer, an ODBO Provider, including MDX engine, and a data store. It does not imply any particular deployment scheme for the components of an ODBO Provider built with the SDK. The Sample ODBO Provider, included with SimbaProvider SDK, integrates all components into client-side COM DLLs, but this is by no means the only possible configuration. Another possibility is for the MDX engine and ISM layer to reside on the database server, while the ISO (Interface Simba OLAP) layer is split between the client and server.

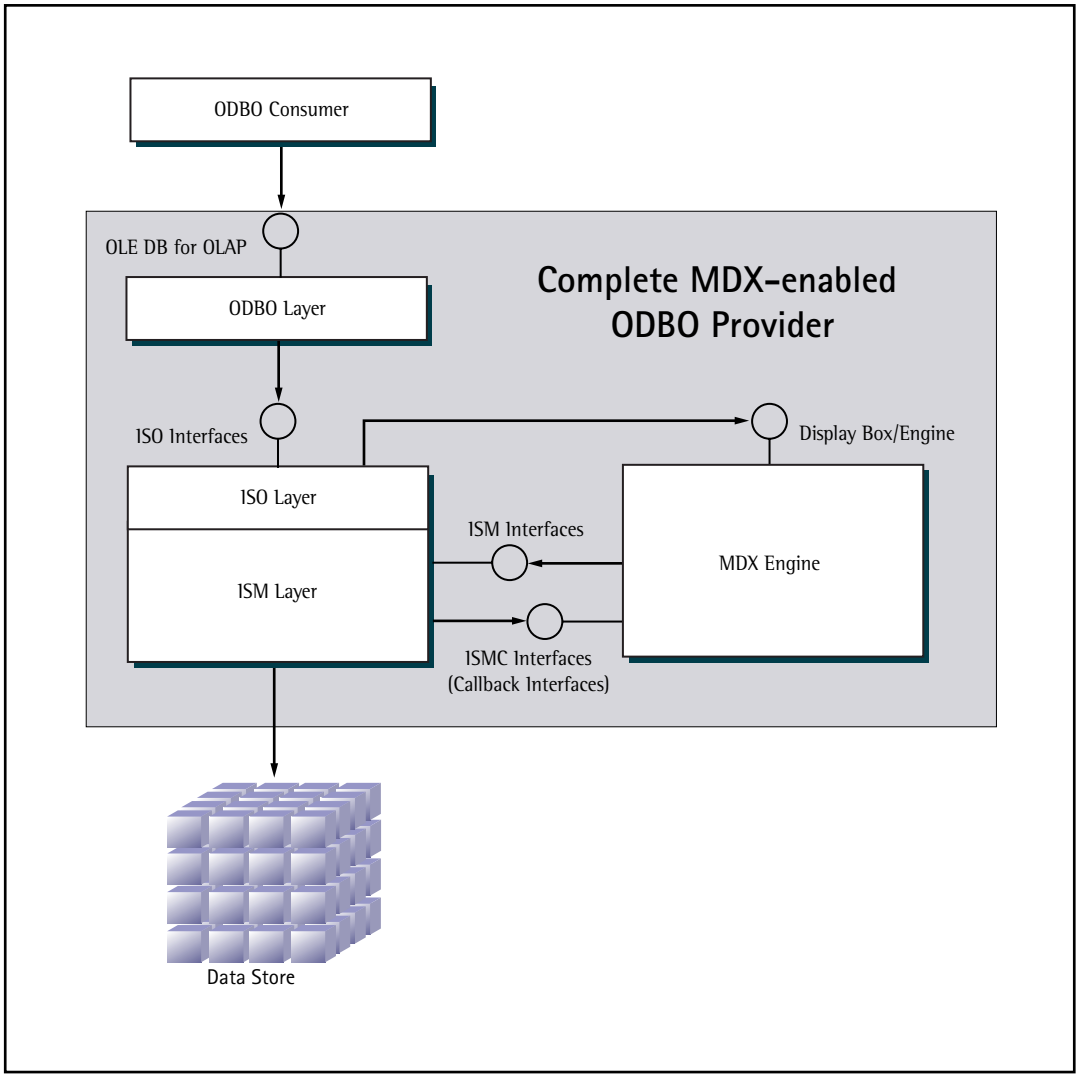

Figure 2. ODBO Architecture

XML for Analysis (XMLA) Architecture

Figures 3 and 4 show the conceptual relationship between a XMLA Consumer, a XMLA Provider, including MDX Engine, and a data store. It does not imply any particular deployment scheme for the components of a Provider built with the SDK. The Sample XMLA Provider, included with SimbaProvider SDK, integrates all components that execute on the web tier; but as with ODBO, other configurations are possible.

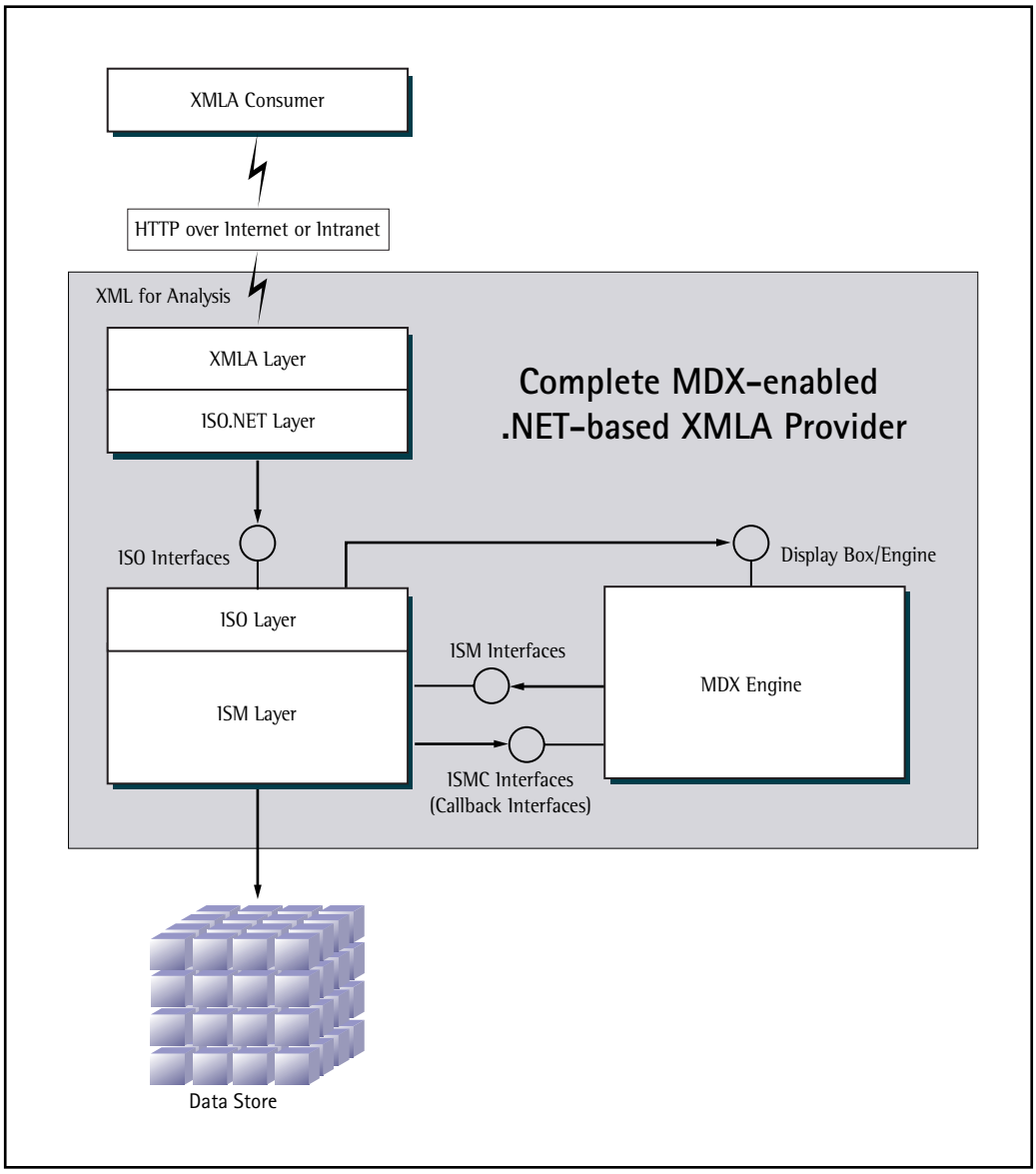

Figure 3. Overview of .NET-based XMLA Provider

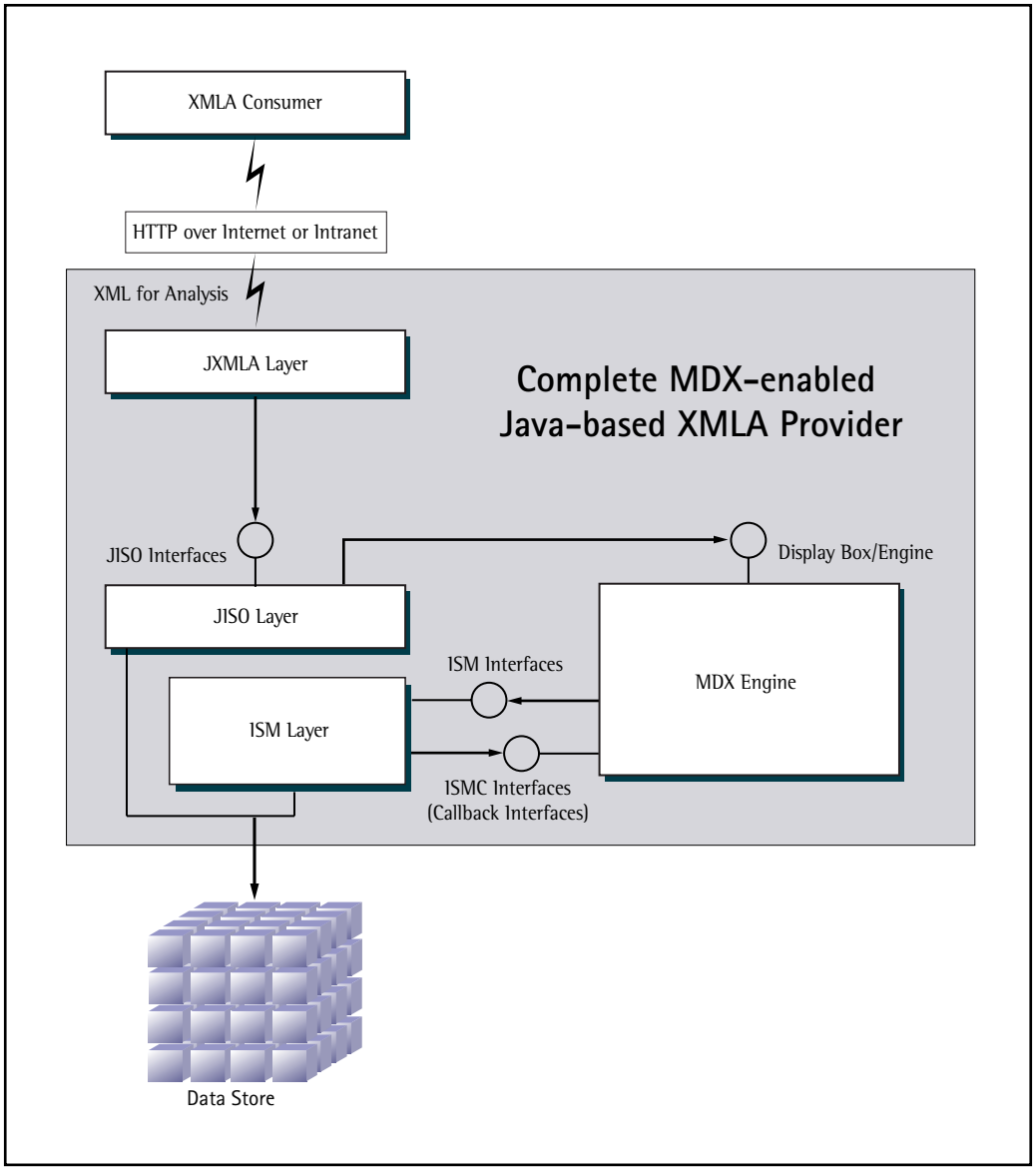

Figure 4. Overview of Java-based XMLA Provider

The following explains the different components within the architecture.

OLE DB for OLAP (ODBO) Consumer

This is an ODBO-compliant application that you wish to provide your data to like SAP Business Objects, IBM Cognos, Microsoft Excel or Microsoft Reporting Services.

XML for Analysis (XMLA) Consumer

This is an application or a component that uses the XMLA protocol[1] to access your data provider over the Internet or an intranet.

[1] We use the term "protocol" here to emphasize that the interaction between the Consumer and Provider follows the model of a communication protocol, rather than a set of call-level interfaces, as is the case in OLE DB for OLAP (ODBO).

ODBO Layer

This component is implemented and maintained by Simba. The ODBO layer calls your ISO layer to access your data store and provides this information in a way that the Consumer will understand. The ODBO classes make no assumptions about the naming of the your ISO classes, so the derived classes can be named whatever you wish.

XMLA Layer

This component is implemented and maintained by Simba. The XMLA layer forms the core functionality of an ASP.NET Web Service that implements the XMLA specification. It uses your implementation layers indirectly, via the ISO.NET layer. You create your own XMLA Provider by sub classing the **Simba.Xmla.XmlaWebService** class, the central component of the XMLA layer.

JXMLA Layer

This component is implemented and maintained by Simba. The JXMLA layer forms the core functionality of a J2EE web service that implements the XMLA specification.

ISO.NET Layer

This component is implemented and maintained by Simba. The ISO.NET layer calls your ISO layer to access your data store. It acts as a bridge between the managed .NET runtime and your unmanaged C++ implementation classes. This layer also implements a connection pooling system to reuse existing database connections where possible, thereby improving your XMLA Provider's scalability.

ISO Interfaces

These are a set of interfaces that reflect the core functionality of the ODBO and XMLA specifications. These interfaces allow for interaction between the ODBO/XMLA layers and the data store via the ISO layer. In concrete terms, the ISO interfaces are a collection of abstract C++ classes that you will need to derive from and implement. Your implementation of these interfaces against your data store constitutes the ISO layer.

JISO Interfaces

These are a set of interfaces that reflect the core functionality of the ODBO and XMLA specifications. These interfaces allow for interaction between the ODBO/XMLA layers and the data store via the JISO layer. The JISO interfaces are a collection of Java interfaces that you will need to implement. Your implementation of these interfaces constitutes the JISO layer.

ISO Layer

This component consists of C++ code that you will write. It implements the ISO interfaces to access your data source. The primary responsibilities of your ISO layer are connecting/disconnecting from the data source, session management, command execution, and formatting your proprietary data and metadata into a tabular form suitable for consumption by the ODBO/XMLA layers.

JISO Layer

This component consists of Java code that you will write. It implements the JISO interfaces so that the ODBO/XMLA layers can query for data and metadata. The primary responsibilities of your JISO layer are connecting/disconnecting from the data source, session management, command execution and formatting your proprietary data and metadata into a tabular form suitable for consumption by the ODBO/XMLA layers.

MDX Engine

This component is implemented and maintained by Simba. The MDX engine component parses MDX queries, type-checks identifiers, evaluates axes and cells, and returns the resulting dataset to the ISO layer for presentation to the Consumer via the ODBO/XMLA layers. The MDX engine makes calls to your ISM layer to query your data store's data and metadata. Simba's MDX engine supports the latest MDX 2005 query language enhancements.

ODBO/XMLA layers

The MDX engine makes calls to your ISM layer to query your data store's data and metadata.

ISM Interfaces

These interfaces allow the MDX engine to request information from the ISM layer.

ISMC Interfaces

These are MDX engine interfaces that allow the ISM layer to call back to the MDX engine component.

ISM Layer

This component consists of C++ code that you will write. It implements the ISM interfaces to facilitate communication with the MDX engine. The primary responsibilities of your ISM layer are metadata caching and traversal, name resolution for MDX language identifiers, MDX function/method implementation, and retrieving cell data from your data store.

DisplayBox/Engine

This is not an interface, although it is drawn like an interface in Figures 2 and 3. The DisplayBox is an object that returns the results of the processed MDX query to the ISO layer. The ISO layer reconfigures the results, so they will meet ISO interface standards, and then returns the data to the ODBO/XMLA layer for presentation to the Consumer.

Data Store

This is your data in its native format and your multi-dimensional application programmer's interface (API). The complete data provider makes calls to your API to retrieve data and metadata.

Utils

Utilities are not shown in the diagram because it is not really a "layer." Rather, it is a class library that you are encouraged to use when developing your ISO and ISM layers. It includes many services, such as Unicode string handling and conversion, date and time manipulation and formatting, auto-pointers for easier memory management, and utilities for threading and synchronization. Everything in Utils has been implemented to make porting your data provider to other platforms easier. This may be an important factor, if you wish to deploy the MDX engine on a Linux server, for example.

Specifications

The SimbaProvider for OLAP SDK requires:

- 1 GHz Windows PC
- 2GB RAM minimum, 4GB RAM recommended
- Windows XP Professional SP3, Windows Vista SP1, Windows 7, Windows Server 2003 R2, Windows Server 2008, or Linux
- 1GB of disk space (depending on build configurations; not including disk space requirements for third-party software)
- Microsoft Visual Studio .NET 2010 SP1 or 2008 SP1; or Linux GCC Compiler
- Microsoft Analysis Services 2008 (for testing purposes)
- Microsoft Excel 2010, 2007 and/or 2003 (for testing purposes)

If you plan to develop in Java, you will also need:

- JDK 1.6 SDK
- Eclipse IDE 3.3 or higher
- JBoss 4.0 or higher

Building a Data Provider

Your first task is to decide whether you wish to build a 32-bit or 64-bit OLE DB for OLAP (ODBO) or a XML for Analysis (XMLA) Provider. Although SimbaProvider SDK supports both ODBO and XMLA, you only need to write a single implementation layer that can be used in either type of data provider. A summary of the ten steps required to develop your data provider on the Windows or Linux platform is included below. Simba provides complete details and documentation in your SimbaProvider Evaluation SDK.

Step One: Connecting to and disconnecting from your data source

Using the Sample ISO Implementation, you must modify the ISODatasource interface in the ISOImpl project.

Modify the following methods:

1. **Datasource:Connect** – You need to implement this method so that it establishes a connection to your data source.
2. **Datasource: SetCurrentCatalog** – This method changes the catalog to which the Provider is currently connected.
3. **Datasource's Destructor** – You need to implement this method so that it disconnects from your data source.
4. **Datasource: Reset** – This method is required to support connection pooling for your Provider.

Step Two: Mapping your metadata to schema rowsets

Metadata in ODBO and XMLA is exposed to consumer applications in a tabular form called schema rowsets. In ODBO, this is done via COM objects, whereas XMLA uses XML documents. In SimbaProvider SDK, simpler objects called schema tables are used to implement schema rowsets.

SimbaProvider SDK automatically converts schema tables to the COM objects, representing schema rowsets for your ODBO Provider, and to XML documents for your XMLA Provider. To implement schema tables, you must retrieve metadata from your data source and expose it in objects that implement the ISOTable interface.

Before you can implement the retrieval of metadata from your data source, you need to determine how it maps to the schema rowsets that you are required to support. You may also have other schema rowsets that you want your data provider to expose. This work requires consideration of the following:

- The ODBO/XMLA metadata model
- Shared dimensions
- Multiple hierarchies
- Unconventional hierarchy structures
- Level support

Step Three: Retrieving metadata from your data source

Using the Sample ISO Implementation to develop your own data provider, you must modify the ISOSession implementation in the ISOImpl project. This step requires modification of the following methods:

1. **Datasource: GetInfo** – Use the Sample ISO Implementation as a starting point.
2. **Session: Constructor** – Initialize any member variables here; you may need to pass other parameters to the constructor, depending on your specific implementation.
3. **Session: Connect** – Remove the **DataStoreAPI: DataStoreCreateSession** call. Your implementation of **Session: Connect** needs to establish a session with your data store and maintain the session state in the session object.

4. **Session: Destructor** – Replace the **DataStoreAPI: DataStoreDestroySession** call with appropriate call(s) to your own data store to destroy the session.
5. **Session: CreateSchemaTable** – This method is the "central switchboard" for retrieving metadata in the Sample ISO Implementation.

Step Four: Exposing metadata as schema rowsets

Using the Sample ISO Implementation to develop your own data provider, you must modify the ISOTable and ISORow implementations in the ISOImpl project.

The ISOTable interface is an abstraction of a collection of cells organized into rows and columns. Cells are represented by the ISOCell interface, rows by the ISORow interface, and column metadata by the ISOColumnInfo interface. The structure of a row in an ISOTable is specified by an ISOColumnInfoVector that contains one ISOColumnInfo for each column in the table. The table itself produces rows that implement the ISORow interface. Each row contains a collection of ISOCells, one corresponding to each ISOColumnInfo of the table.

Step 5: Implementing NameSpecs

ISMNameSpec and **ISMNameSpecFactory** are both involved in the resolution phases of MDX expression processing. After parsing a MDX statement, the MDX engine inspects the statement to find names. In this context, names are sequences of identifiers that are not keywords, functions, methods, or flags. For example, [Product].[Beer and Wine].[Beer] is a name; CrossJoin, SELECT, DIMENSION PROPERTIES, etc. are not names. Each time the MDX engine isolates a name, it needs an object to encapsulate that name. The interface defined for these objects is **ISMNameSpec**. The MDX engine requests a new **ISMNameSpec** object through the **ISMNameSpecFactory** interface.

Step 6: Building entities

Entity objects are abstract representations of metadata like cubes, dimensions, hierarchies, levels, members, etc. Entity objects also represent properties, values and cells. Entities are all related via an interface hierarchy. This hierarchy allows the MDX engine to obtain metadata, discover its type, build axis sets and slicers, locate cells, and encapsulate calculations (like those in a WITH clause). The MDX engine uses Entities to:

- Validate the statement
- Encapsulate calculations such as those in the WITH clause or the CREATE statement
- Determine axes of the query, including the slicer
- Specify locations in the cube for which cell data is needed
- Move and manipulate cell data

Step 7: Implementing the Resolver component

The Resolver component is responsible for handling **NameSpec** objects and identifying whether a given **NameSpec** refers to a member, dimension, level, or hierarchy. The Resolver also resolves cubes, functions, methods, constant values, and cell and dimension properties. The MDX engine uses the Resolver, via the **ISMResolution** interface, to convert **NameSpecs** into objects that represent entities within your metadata.

Step 8: Implementing the Evaluator component

Once the MDX engine has resolved all the NameSpecs within a MDX statement into entities, it begins evaluation. Evaluation is the process of transforming resolved expressions into entities that represent the results of those expressions. For a SELECT statement, these result entities include the axis sets, slicer tuple and cells. During evaluation, the MDX engine will call your Evaluator component via the **ISMEvaluation** interface. Your Evaluator component is responsible for applying cell and dimension properties, customizing the MDX engine's cell-evaluation behavior, and fetching cells from your database.

Step 9: Building the dataset

In order to present query results to OLAP Consumers, your Provider's ISO layer must implement the dataset object. A dataset is a representation of the **DisplayBox** that the ODBO/XMLA layers can call via the **ISODataset** interface. The **DisplayBox** is assembled by the MDX engine from the slicer tuple and axis sets, created during the evaluation phase. It is populated with cell data on demand.

Step 10: Executing a MDX query

The **ISOCommand** interface is used to execute commands and create datasets. You can reuse the existing **ISOCommand** implementation for executing queries. You may wish to modify the Execute command to create a dataset or flattened rowset that is derived from **ISOResultType**.

Summary

With SimbaProvider SDK, you can build a functional ODBO or XMLA Provider in about a month, and have a production quality data provider ready to ship within six to 12 person months, depending on the complexity of your data source. When you license SimbaProvider SDK, you get the ability to develop both an ODBO and a XMLA Provider using the same toolkit. You can develop your data provider on Windows XP, Windows Vista, Windows 7, or Linux. You build one provider and have the choice of interfaces offered to client applications – ODBO and/or XMLA.

About Simba Technologies Inc.

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

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

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