Paper 136

What's Up with OLE DB?

Thomas W. Cox, SAS Institute Inc., Cary, NC

ABSTRACT

During the early 90's, Microsoft promoted ODBC as the standard API for accessing relational/tabular data on the Windows® Platform. Today, Microsoft is advancing a new component based data access framework, OLE DB; which extends a common data access framework to a wider variety of data sources, including non-relational and multi-dimensional stores. This paper will give an overview of OLE DB fundamentals and related technologies included under Microsoft's Universal Data Access umbrella. In addition, it will preview products that SAS Institute is developing to leverage the capabilities of OLE DB and SAS® software.

INTRODUCTION

A key aspect of SAS Institute's Nashville project is the "Intelligent Client," a concept that embraces an "…open architecture that allows third-party software to tap into the power of SAS servers." [SASC98] The Institute has a long history of providing access to external data and exposing SAS data sets to external sources via SAS/Access® software, but the Nashville project brings an increased commitment to openness and ease of use in sharing data between SAS software and other systems.

The software industry at large has also seen a shift to more open data exchange; driven by the ease and low cost of PC data analysis and reporting tools, the proliferation of data warehousing, and the Internet. These technologies, among other factors, have created the need for a common method of data access for a variety of data, regardless of its origin.

OLE DB is one of the most successful attempts to address that need. In the sections that follow, we will look at how OLE DB came to be, how it works, what we can expect from it in the near future, and most importantly, how it is helping SAS Institute and our customers create better information delivery solutions.

OLE DB

WHAT IS OLE DB?

OLE DB is not a product – at least in the traditional sense. Instead, it is a specification of how data *consumers*[1] should communicate with data *providers*. The goal of OLE DB is to provide an open[2], extensible standard; allowing any software that conforms to the specification to communicate via a set of common *interfaces*.

OLE DB is built on some of Microsoft's key strategic technologies, and builds upon the heritage of the previous generation of data access software. In order to understand more clearly the reasoning that underlies OLE DB, and how those strategic technologies affect the entire market, we will need to review the history of two of the key developments that led to the creation of OLE DB.

A SHORT HISTORY OF DATA ACCESS

In the bad old days, (circa 1984-1991,) if you needed a program to access data source specific data from a third-party data location - whether it was an *RDBMS*, spreadsheet, or legacy system – a great deal of effort and code was required.

For Microsoft Excel®, it was possible to buy the file format specification and write an extensive parsing tool, but it was usually easier to export it to some text-based format and re-format it manually. Embedded *SQL* and clunky preprocessors were the tools of choice to access early RDBMS servers. Sybase provided one of the best options at the time with its DBLIB database API, but it suffered from most of the same drawbacks as the other approaches: a proprietary design, limited flexibility, etc.[3] At least one major independent software vendor was founded to provide tools to address the problem; and thrived by selling what were, unfortunately, only stop-gap relief measures.

Recognizing these problems, the SQL Access Group (SAG) was formed in 1989 as an industry consortium to promote portability and interoperability among DBMS vendors. Microsoft representatives were among the members, and the company anticipated the eventual release of the SAG standard by shipping the Open Database Connectivity API (*ODBC*) and tool-set in 1992.[4]

ODBC was a major step forward in data connectivity, and played a significant role in the legitimization of the PC as a client platform for enterprise applications. It also fueled the sales of PC-based report-writers, rapid application development (RAD) environments, and related tools.

ODBC has its share of problems, however. As a product of SAG, ODBC is tightly bound to the SQL language, and therefore to the relational model that underlies it. This creates a problem when attempting to use ODBC to access hierarchical, multi-dimensional (*OLAP*), or free-form (spreadsheets, e-mail) data. The requirement for

---

[1] Acronyms, technical terms, and common words used in a specific technical sense are initially presented in italic type, and are defined in the Glossary of Technical terms at the end of the paper.

[2] OLE DB is also portable, at least to the extent that it will run on any hardware platform for which COM is available, however OLE DB at this time is clearly Windows-centric, an aspect you will find reflected in this paper.

[3] For a more extensive review of these issues, and data access history as well, refer to Chapter One of "Inside ODBC" [Geiger]

[4] ODBC and the X/Open SQL Command Language Interface standard written by SAG were re-aligned in 1996 with the release of ODBC 3.0

SQL processing also poses an entry barrier for many of the low-end tabular data stores.

Another problem is the "leveling" of ODBC drivers. The designers of ODBC anticipated that not all data-sources would be capable of providing the full functionality, so the API entries were grouped into "Core", "Level 1", and "Level 2" to indicate which features were required and which considered optional for drivers of various capabilities. Unfortunately, since the development of most drivers is outside Microsoft's control, many use a "pick-and-choose" approach to ODBC API support.

Similarly, the many flavors of SQL result in each driver having to report its degree of conformance to various SQL specifications. The resulting complexity of these issues, exacerbated by the changes contained in various versions of ODBC, seriously undermines the universal nature of the API. In the worst-case scenarios, developers are forced to treat each DBMS separately – just as they did before ODBC was available.

To complicate matters further, other software groups at Microsoft were proceeding to fragment the database middleware arena. Special-purpose APIs such as Data Access Objects (DAO) and Remote Data Objects (RDO) were confusing developers and competing for limited development and support resources.

Finally, the deployment of ODBC is in many cases a logistical nightmare, requiring specific versions of the server, driver, network transport, operating system, and client software to be compatible in order for the system to work. This issue is not unique to ODBC, however; and Microsoft already has a mechanism to address it: COM.

So, in order to deal with this stack of problems (and to address the increasingly important issues of Internet integration, multi-threaded applications, and other technical issues) Microsoft developed the OLE DB specification. This paper will later discuss how it addresses the data-specific issues later, but we first need an understanding of the technologies referred to variously as OLE, ActiveX, and COM.

A BRIEF OVERVIEW OF COM

Software developers have been talking and writing about the "software crisis" since the late 1960's. While that is too complex an issue to discuss here, in essence it refers to the fact that software productivity and quality continues to fall further and further behind hardware quality and capabilities. One partial solution that has been applied to this situation is software component reuse: in the same way that a computer designer creates a system from a set of common parts, software developers would like to be able to build a system from reusable software components.

Object-oriented software languages and design have contributed to this effort, and defined some useful concepts for managing reuse, such as *encapsulation, polymorphism*, and *inheritance*. Among other benefits, these concepts all serve to isolate changes in one piece of software, preventing them from affecting other software.

COM uses these object-oriented concepts, but unlike most other approaches, COM attempts to promote software component reuse at the binary object level instead of at the code level. This has several benefits, but most importantly, it allows a software component to be upgraded or replaced transparently to the applications using it.

The primary mechanism in COM for insuring consistency across versions is the interface, and the first rule of COM programming is that any interface, once published, is immutable. Specifically, the syntax of all methods and their arguments must remain the same[1]. In addition, all access to a com object must be though the interface methods: directly modifying an object's data is forbidden (encapsulation.)

As an example, if you think of a COM object that provides a specific functionality – say a drawing tool – it might have an interface called IGeometry[2], including methods such as DrawCircle(center, radius) and DrawRectangle(center, height, width). It would probably also have other interfaces such as IPaint, IText, etc.

Now suppose you want to update the drawing object to allow specification of a rectangle by the upper left and lower right vertices. In the COM framework, it is not permissible to change the parameters of the DrawRectangle() method to be (upper_left, lower_right). Further, it is not allowed to add a DrawRectangle2() method to the IGeometry interface. Instead, you could provide an entirely new interface, perhaps called IPointGeometry.

Another key aspect of the interface is that it is not tied to a specific object, but can be reused (inheritance). This allows the calling program to treat all objects that provide a specific interface identically (polymorphism). To further leverage this capability, COM provides a mechanism for a program to determine which interfaces each object supports at execution time, eliminating the need for the consumer to limit itself to a lowest-common-denominator functionality.

In addition to addressing the above problems, COM was designed with network computing in mind. Such programming issues as multi-threaded and parallel computing, distributed transaction coordination, were addressed and generic solutions devised. As one example, the COM tradition of passing large complex structures as parameters instead of concise, easy-to-use methods was not a matter of sloppy design, but a conscious trade-off aimed at reducing the number of network packets exchanged, and thereby greatly improving distributed application performance.

---

[1] It must be noted that there are some gaps in COM's implementation of immutability. First, there is no COM police to arrest you if you do change the syntax of a method call – it is just very bad manners. Secondly, there are no explicit restrictions on the *semantics* of a method, so that behavior not constrained by the parameters could change. While this is a necessity, it does leave room for some nasty side effects.

[2] COM Interfaces traditionally begin with 'I'.

Clearly, if a calling program can always rely on the binary compatibility of a method call, and the functionality of an object can be clearly and concisely determined on the fly, it greatly reduces the exposure to program failure when a component is upgraded. COM, therefore, was the logical solution to many of the problems in ODBC.

HOW DOES OLE DB WORK?

From a system perspective, OLE DB is still simply database middleware like ODBC, in that it serves as a translator between a client and a server. The calling application obtains a COM Interface handles and executes the methods of interest.

"Client" and "server" are sufficiently ambiguous terms that OLE DB has introduced the terms "consumer" and "provider" to clarify the relation. A consumer is any software that requests an OLE DB interface, while a provider is any software that implements one or more OLE DB interfaces.

OLE DB differs from ODBC in that there is no equivalent to the ODBC driver manager. OLE DB providers are self sufficient in this sense, although multiple providers may cooperate to access a single data source. For example, Microsoft distributes a provider enumerator (which is itself an OLE DB provider) as part of the OLE DB redistributable files. It is used to obtain the list of installed providers.

An important aspect of OLE DB is that instead of entirely replacing ODBC, it leverages the existing base of ODBC drivers by providing a translation provider – the "OLE DB Provider for ODBC Data Sources" for those sources.

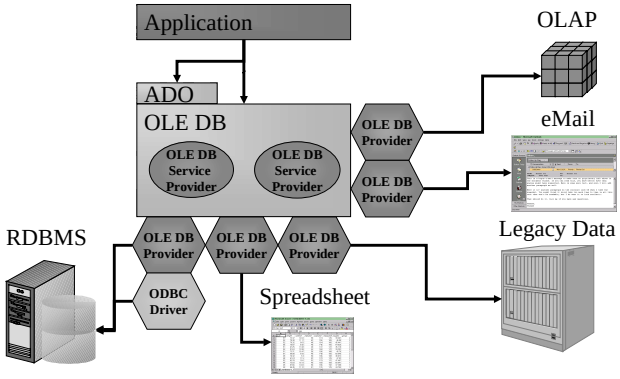

Figure 1: OLE DB Context

The OLE DB API, like all COM APIs, is technically language independent; but it is most conveniently accessed in C++. To make OLE DB technology more accessible to languages such as Visual Basic, Microsoft has provided an abstraction layer called Active Data Objects (ADO). ADO provides most of the power of OLE DB, but with less code and a much shallower learning curve.

OLE DB Objects

While an in-depth review of the OLE DB API is outside the scope of this paper, the basic architecture deserves a closer look. There are seven object types in OLE DB:

Table 1: OLE DB Objects

Enumerators	Provide a list of installed data sources, other enumerators, or similar information.
Data Sources	Provide connections to a DBMS, file, etc.
Sessions	Provide a space for transactions.
Transactions	Commit or abort database changes.
Commands	Execute an SQL (or other data source language) command.
Rowsets	Provide tabular data.
Errors	Provide database error information.

Each object supports one or more COM interfaces, some of which are part of the standard COM repertoire, but most were created specifically for OLE DB.

The objects are often inter-dependant: for example, a data source object is required in order to create a session object. Note that an application may have multiple data source objects, each with multiple sessions, which in turn may have multiple transactions, commands, and rowsets. (See Figure 2)

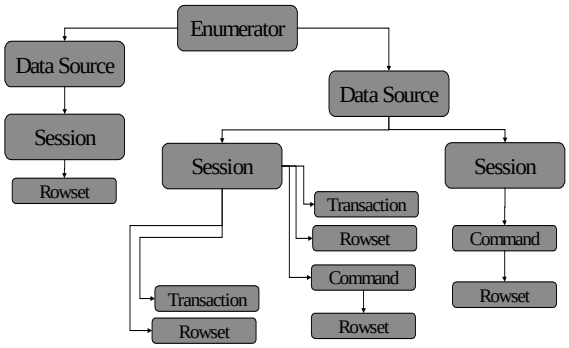

Figure 2: Sample OLE DB Dependencies

Enumerators

Enumerators provide lists of information to the consumer. The most common example, the root provider supplied by Microsoft searches for installed OLE DB providers, including other enumerators, and returns their identifying information in a rowset.

Enumerators are most useful to generic consumers: those that are designed to connect to arbitrary providers selected at run-time. In some cases, however, a data provider will include additional enumerators; such as when an RDBMS manages multiple databases residing on a single server.

Data Sources

Data source objects perform the work required to connect to a data source[1], such as a tabular file or a DBMS. They manage connection options; make the network connection or open the file(s), as appropriate; and may supply

[1] It should be noted that there is a clear distinction between an OLE DB data source object and the actual data source it represents. Throughout this section, the OLE DB component will always be referred to as a data source object. While this terminology overlap is occasionally confusing, the name does clearly convey the role of the object.

administrative capabilities and general information about the data source.

All consumers must obtain a data source object. Further, they must initialize the data source object before it can create a session or perform other operations of the data source.

Sessions

Sessions provide a framework for transaction management, which can be handled automatically, or explicitly controlled by the consumer. If a data source doesn't support transactions, the session object will operate in auto-commit mode, where all changes will be saved as they are made.

Sessions are a prerequisite for creating transactions, commands, and most rowsets. They may also provide schema information and the ability to create or modify tables and indexes.

Transactions

Transaction objects are optional, and serve a single purpose. They allow the consumer to explicitly control the commission or abortion of pending database options. Not all providers support transactions, and those that do may handle only simple transactions. More sophisticated providers may participate in nested or coordinated transaction management.

Commands

Commands are used to submit statements in SQL (or other appropriate language, such as the MDX extended SQL syntax for OLAP) to the provider. Unlike ODBC, command support is optional for OLE DB providers. Those providers that do have command languages may use different dialects, although they must report which dialect(s) are supported, and to what extent.

Commands that produce a result set can create a corresponding rowset object, but SQL commands are not required to retrieve data. There is no mandatory command support, so generic consumers are less likely to rely exclusively on commands as an access mechanism. ANSI SQL is widely enough adopted to make it useful for many purposes, however.

Rowsets

Rowsets are the OLE DB objects that contain the data: the actual rows and columns, or observations and variables if you prefer. A rowset is effectively a local buffer for the data, although advanced providers support a vast array of client and server-side cursors, scrolling, bookmarks, etc. through optional interfaces. This allows buffer management to be tuned effectively by each consumer based on its usage pattern and the capabilities of the provider.

Rowsets can be created from a data source directly, from a command, or in certain cases by an enumerator. Indexes - high-performance rowsets based on an index in the data source – are also supported.

Errors

Errors in OLE DB are simply an extension of the error handling objects built into COM. Support for error objects are, sadly, entirely at the discretion of the provider; often requiring the consumer to rely on basic method return codes to check for success or failure. However, the better providers can and do provide extensive diagnostic messages in the error objects.

All of the other OLE DB objects may create error objects. OLE DB expands upon the basic COM errors by providing for multiple error events in a single error object through the creation of an error record interface, which exposes an array of error information.

WHERE ARE WE NOW?

OLE DB 1.0 was released in the summer of 1996, and like many 1.0 products, it had more value as a proof-of-concept than as a usable tool. Versions 1.1 and 1.5 stabilized and expanded the specification, but still had functionality gaps in the areas of remote data access and security.

Version 2.0, released in the summer of 1998, incorporated the OLE DB for OLAP extensions and was considerably more mature; the first release capable of filling the strategic niche prepared for it. In the coming year, Microsoft and other vendors will be releasing a number of core products that will rely on OLE DB as a primary data access mechanism[1].

NOTABLE OLE DB PROVIDERS AND CONSUMERS

As mentioned earlier, OLE DB is a specification, not a product; and would be of no use without industry support. Fortunately, there is already a wide selection of providers available. There are also a number of consumers, though in general application developers have waited for the providers to ship before committing resources to support OLE DB. This trend is resulting in a wave of OLE DB enabled products planned for a coordinated release with Windows 2000.

Providers

All major RDBMS brands have an available OLE DB provider, though some are native while others rely on middleware solutions. Microsoft SQL Server, Microsoft Access[2], and Oracle® providers are available directly from Microsoft, while MicroFocus[3] sells direct providers for Oracle, Sybase, and INFORMIX, as well as middle-tier access to DB2, Open Ingres, and others. ISG provides a

[1] News flash: at press time, the version 2.1 SDK update had just been released.
[2] It really is an RDBMS, however lightweight.
[3] MicroFocus acquired Intersolv, which had previously acquired Q+E, which began as Pioneer Software.

sophisticated middle-tier solution with access to over 20 data sources[1].

Non-relational providers are also available, and are currently available for various flat file formats (such as dBase) and email systems including Lotus Notes and Microsoft Exchange. File system providers are available for Microsoft Active Directory Services[2] (NT, NetWare, and LDAP) and C-ISAM/D-ISAM, with VSAM slated for release soon.

On the OLAP side, providers for the multi-dimensional servers from Microsoft, Oracle, Red Brick, and Sybase are shipping or in the works. The OLE DB for OLAP specification is relatively recent, so more providers are anticipated in this area over the coming year.

SAS Institute itself is developing four providers, three relational and one OLAP, which will be discussed in more detail below. The SAS/Access Interface to OLE DB has the ability to consume data from all of these providers, enabling SAS software to serve as a central information-processing tool for all your data, regardless of its origin.

Consumers

There are two main categories for OLE DB consumer applications: those that process data from a variety of OLE DB providers, and those that are designed to interact with a single provider. The first group – generic consumers – largely consists of "shrink-wrap" software. The single-source consumers are typically proprietary corporate IS applications, and are often bound directly to the schema of a particular database.

Of the shrink-wrap applications, few are available now, but Microsoft has announced sweeping support in the Microsoft Office 2000 suite. Microsoft Excel 2000, for example, will be able to import data from any OLE DB source, and will generate pivot tables linked directly to OLAP cubes. Similarly, Microsoft Access will support import and view capabilities for OLE DB providers. One application that is available now is Crystal Reports 7, from Seagate software, which can produce reports using data from OLE DB sources.

OLE DB for OLAP consumers will see some exciting developments over the next year. Support initiative have been announced by most of the leading OLAP companies, including BAAN, Brio, Cognos, Hyperion, Knosys, Pilot, and of course Microsoft and SAS Institute.

Among proprietary consumers, the important news is not the applications so much as the development tools available to leverage OLE DB. Microsoft Visual Basic and Visual C++ offer tools and "Wizards" to simplify the development of OLE DB-based applications. Sybase has announced OLE DB support in PowerBuilder Version 7. There are also a number of web technologies such as Microsoft ADO/RDS to use OLE DB technology to deploy web solutions.

With such a diversity of providers and consumers, and OLE DB as the common plumbing to connect them, it is easy to see that there are endless scenarios for advanced data connectivity.

SAS SOFTWARE AS AN OLE DB CONSUMER

As mentioned earlier, SAS Institute's Nashville Project is about creating intelligent information delivery tools. As part of that initiative, there are currently five OLE DB initiatives that are sufficiently advanced to discuss, although other opportunities are always being considered. Of the active projects, five are OLE DB providers for the various means of accessing SAS data sets; the other empowers SAS as a consumer of OLE DB data.

SAS/ACCESS INTERFACE TO OLE DB

The SAS/Access Interface to OLE DB is the Institute's all-purpose consumer. It is surprisingly powerful and easy to use. If you can create a connection to an OLE DB data source, you can treat all of the data in that source as if it were a normal SAS data set, but one that maintains a live connection to the external data.

Availability

The OLE DB Interface shipped with SAS/ACCESS, version 7, as an experimental read-only data source. In version 8, it will be a production interface with write capabilities.

Features

This interface is a full libname engine. If you are not familiar with term, libname engines are another exciting part of the SAS Nashville project. Here is a brief description:

> "New in Version 7, dynamic DBMS engines enable the SAS user to assign a SAS library that dynamically connects to the DBMS data server. SAS programs using the library can then directly list, read, update, delete and create DBMS tables. The creation of static access and view descriptors is no longer required." [SASP23][3]

A key feature of libname engines is view creation, supported in this interface by maintaining OLE DB connection information across SAS sessions. These views are especially powerful when combined with the pass-through facility, which allows SQL queries to be processed by the server, greatly increasing the efficiency of the data access.

The SAS/ACCESS Interface to OLE DB supports the OLE DB for OLAP extensions with pass-through MDX command capabilities, when supported by the corresponding provider. MDX is a powerul language for

[1] See http://www.microsoft.com/data/oledb/products/product.htm for a more detailed provider list.
[2] Will not be fully functional until Windows 2000.

[3] The paper quoted here was presented at SUGI 23, and is available in the Proceedings. I highly recommend it for those interested in more detail regarding new data access features.

data subsetting and multi-dimensional OLAP data retrieval.[1] (See Example 3)

Examples

SAS software and OLE DB provide such flexible opportunities for data access that it is impossible to cover all the aspects in the space available. However, here are three examples that demonstrate some of the key features.

First, here is a simple libname statement, to illustrate just how easy it is to create a transparent connection to a SQL Server 7 database. Other connection properties are available, as needed by the specified provider.

Example 1: Creating an OLE DB Libname

```
LIBNAME SQLSrv7 OLEDB
    PROVIDER="SQLOLEDB"
    PROPERTIES=(
        "user id"=dbitest
        password=XXXXXXXX
        "data source"=dbipc6
    );

PROC DATASETS LIB=SQLSrv7;
RUN;

PROC CONTENTS DATA=SQLSrv7.EMPLOYEES;
RUN;
```

Figure 3 shows the results after the execution of Example 1. Each of the available data sets shown in the SAS explorer is a SQL Server table. As you will note in the log and output windows, the new library and its members may be used in PROC statements as if they were native SAS data sets.

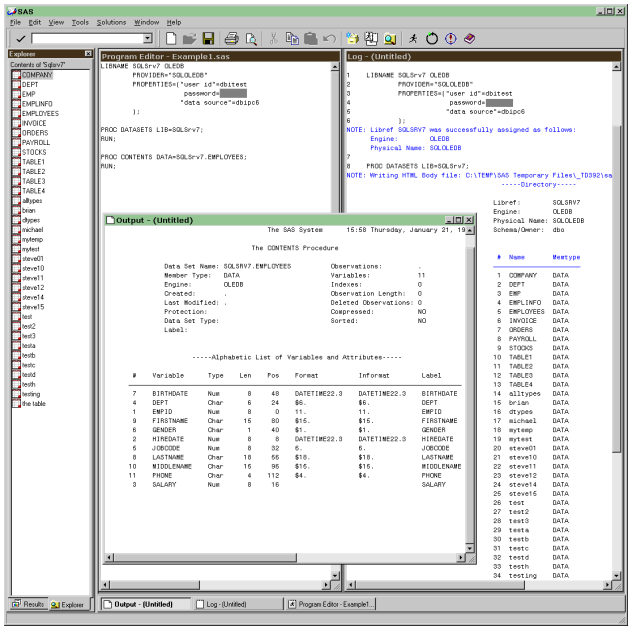

Figure 3: OLE DB Libname Results

Next is a sample that shows two capabilities, SQL pass-through and view creation:

Example 2: OLE DB View Creation and SQL Pass-through

```
PROC SQL;
    CONNECT TO OLEDB (
        PROVIDER="SQLOLEDB"
        PROPERTIES=(
            "user id"=dbitest
            password=XXXXXXXX
            "data source"=dbipc6
        )
    );
    CREATE VIEW SASUSER.V_SS7 AS
        SELECT * FROM CONNECTION TO OLEDB (
            select DEPT,
                   AVG(SALARY) AS "Avg_Salary",
                   COUNT(*) AS "Num_Employees"
            from EMPLOYEES
            where SALARY > 0
            group by DEPT
        );
QUIT;

PROC PRINT DATA=SASUSER.V_SS7;
    FORMAT AVG_SALARY DOLLAR8. ;
RUN;
```

This view will persist across SAS sessions. Output is routed to HTML[2] for this example (see Figure 4.) The sub-setting and aggregation operations are performed natively on the RDBMS server, greatly reducing network traffic and SAS processing time.

Results Viewer - SAS Output

The SAS System

Obs	DEPT	Avg_Salary	Num_Employees
1	ACC013	$54,591	3
2	ACC024	$55,371	3
3	ACC043	$75,000	1
4	CSR004	$17,000	1
5	CSR010	$44,324	4
6	CSR011	$41,966	2
7	SHP002	$40,111	3
8	SHP013	$41,068	3
9	SHP024	$50,000	1

Figure 4: Output from an OLE DB View

The final example of SAS software as an OLE DB consumer will demonstrate OLAP capabilities by using the MDX command pass-through feature. This query aggregates results along two of the multi-dimensional axes into a summary report.

Example 3: MDX Pass-through

```
PROC SQL;
    CONNECT TO OLEDB (
        PROVIDER=msolap
        PROPERTIES=(
            'user id'=dbitest
            password=XXXXXXXX
            'data source'=dbipc6
        )
    );
    SELECT * FROM CONNECTION TO OLEDB (
        MDX::select {[ Measures].[Unit Sales]} on columns,
        order(except([Promotion Media].[Media Type].members,
        {[Promotion Media].[Media Type].[No Media]}),
```

[1] At press time, the OLAP capabilities will be considered experimental in version 8, with MDX command results limited to two axes per query.

[2] HTML output is an option that may be selected at any time via the preferences dialog.

```
        [Measures].[Unit Sales],DESC) on rows
        from Sales
    );
QUIT;
```

For this example, the results of PROC SQL are simply routed to the output window.

Promotion Media	Unit Sales
Daily Paper, Radio, TV	9,513.0
Daily Paper	7,738.0
Product Attachment	7,544.0
Daily Paper, Radio	6,891.0
Cash Register Handout	6,697.0
Sunday Paper, Radio	5,945.0
Street Handout	5,753.0
Sunday Paper	4,339.0
Bulk Mail	4,320.0
In-Store Coupon	3,798.0
TV	3,607.0
Sunday Paper, Radio, TV	2,726.0
Radio	2,454.0

Figure 5: MDX Query Results

The usage scenarios for the SAS/Access Interface to OLE DB are endless. If there is an OLE DB provider or ODBC driver for the data source you wish to access, this engine can deliver it.

SAS OLE DB PROVIDERS

Three of the five SAS OLE DB providers are designed to deliver SAS data sets to consumers. They are distinguished largely by the source of the data they provide: local SAS data sets, data from SAS/SHARE® servers, or data from the new SAS RIO servers.

SAS OLE DB BASEPROVIDER

The first - BaseProvider - is a lightweight provider of SAS data sets residing on the local Windows system, or within the extended network file system. It allows the consumer to read and modify data sets directly without the overhead of Base SAS. It supports V6, V7 and V8 SAS software for Windows data set file formats.

Availability

The SAS OLE DB BaseProvider will ship as an add-on product, using a distribution mechanism similar to that of the SAS Universal ODBC driver.[1]

Features

BaseProvider supports both read-only and update access to SAS data set files. Files are opened for exclusive access. In other words, only one OLE DB rowset may be opened on a data set at a time, likewise, the data set may not be in use by SAS software and OLE DB at the same time. The provider implements both immediate and delayed update modes supported by OLE DB. This effectively makes the BaseProvider a single-user solution.

OLE DB's random access services are also implemented, enabling cursors. The BaseProvider passes Microsoft conformance tests and is compatible with ADO 2.0.

Example

Here is the complete HTML source for a lightweight FSBrowse-style web interface to SAS data sets, written in VBScript, and using the ADO wrapper classes for OLE DB:

Example 4[2]: Code for Web Interface to SAS Data Sets via ADO

```
<html>
<head><title>ActiveX Data Objects (ADO)</title><head>
<body background="sas8bg.jpg" BGCOLOR=ffffcc TEXT="#000000">

<h1>SAS Data Sets via ADO</h1>

This page automatically generates the data form based on the
schema information found in the recordset (SASHELP.SHOES) using
VBScript, ADO and SAS OLE DB BaseProvider. <P>

<input type=button name="btnFirst" value="  <<  " >
<input type=button name="btnPrev" value="  <  " >
<input type=button name="btnNext" value="  >  " >
<input type=button name="btnLast" value="  >>  " >
<br><br>
<input type=button name="btnGoto" value=" Go To " >
<input type=text size=5  name=inpObsNum value="1">
<br>

<hr>

<!-- ADO recordset object -->
<object id=rs classid="clsid:00000535-0000-0010-8000-
00AA006D2EA4">
</object>

<script language="VBScript">

   '*************************************************
   '**  Open the recordset / execute the sql query
   rs.CursorType = adOpenStatic
   rs.CursorLocation = 3 '** adUseClient
   rs.Open "shoes", "Provider=sasafbas.DataSource.1;" &_
      "Data Source=C:\Program Files\SASm800\core\sashelp", _
      ,, adCmdTable

   set flds = rs.Fields
   '*************************************************
   '**  Build the html for viewing the data.
   '**  1: determine width for field-name column
   namewid = 12
   for i = 0 to flds.Count - 1
      if len(flds(i).Name) > namewid then
         namewid = len(flds(i).Name)
      end if
   next

   '** 2: write out html for form-fields
   document.write "<pre><B>"
   s =  "Observation"
   s = s + space( namewid - len(s) + 2 )
   s = s + "<input type=text name=fld_obs>"
   document.write s
   document.write "</B></pre>"
   for i = 0 to flds.Count - 1
      document.write "<pre><B>"
      s = flds(i).Name
       s = s + space( namewid - len(s) + 2 )
       s = s + "<input type=text name=fld_" + cstr(i) + ">"
       document.write s
       document.write "</B></pre>"
   next

   '*************************************************
   '**  Build the vbscript for copying data to the form

   document.writeln "<script language=""vbscript"">"

   document.writeln "sub FillForm"
   document.writeln "fld_obs.value = rs.AbsolutePosition & _
   ""of "" & rs.RecordCount"
   for i = 0 to flds.Count - 1
      s = "fld_" + cstr(i) + ".value = " + "rs.fields(" + _
          cstr(i) +  ").value"
      document.writeln s
   next
   document.writeln "end sub"

   '** force 'FillForm' to execute immediately
   document.writeln "FillForm"
   document.writeln "</" + "script>"
</script>
```

[1] At press time, pricing, beta schedules and target release date were not available.

[2] This example is derived from some a sample included with the Microsoft Data Access SDK 2.0.

```
<script language="VBScript">
   Sub btnNext_OnClick
   if not rs.EOF then
      rs.MoveNext
      if rs.EOF then
         rs.MoveLast
      else
         FillForm
      end if
   end if
   End Sub

   Sub btnPrev_OnClick
   if not rs.BOF then
      rs.MovePrevious
      if rs.BOF then
         rs.MoveFirst
      else
         FillForm
      end if
   end if
   End Sub

   Sub btnFirst_OnClick
   rs.MoveFirst
   FillForm
   End Sub

   Sub btnLast_OnClick
   rs.MoveLast
   FillForm
   End Sub

   Sub btnGoto_OnClick
       x = CInt(inpObsNum.value)
       if x < 1 or x > rs.RecordCount then
           MsgBox "Observation " & x & " is out of range."
       else
           rs.Move x - 1, 1 '** adBookmarkFirst
           FillForm
       end if
   End Sub
</script>
<hr>
<img src="ie_anim.gif" width="88" height="31" align=right
alt="ie">

</body>
</html>
```

Figure 6 shows the resulting web page, as displayed in Microsoft Internet Explorer, version 4.01.

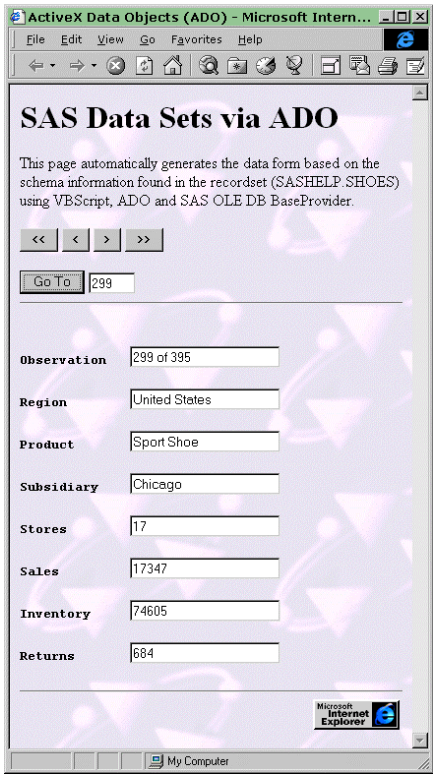

Figure 6: Web Interface to SAS Data Sets via ADO

SAS IOM SERVER OLE DB RIO PROVIDER

As part of the SAS Nashville project, the Institute is developing a set of interfaces to expose SAS libraries and files to automation server clients. Collectively, these interfaces are referred to as RIO (Remote I/O).

RIO is designed to use the Interface Object Model (IOM,) which defines a structure for distributed object programming in SAS software and for standalone components. IOM supplies an object-oriented programming discipline for SAS components that are available to non-SAS clients which conform to the standard COM and CORBA distributed object architectures.

RIO is designed to transfer SAS data efficiently and accurately between client and server contexts. The RIO OLE DB provider hides the complexities of RIO from client applications, and opens SAS RIO servers to access by generic consumers.

Availability

The SAS OLE DB RIOProvider will ship with the SAS IOM server as an optional component.[1]

Features

RIOProvider supports both read-only and update access to data files. It also allows the creation of new data sets. Further, it enables OLE DB command object capabilities by providing SQL support.

RIOProvider implements OLE DB's random access services, including the IRowsetLocate and IRowsetScroll interfaces, enabling server-side cursors. RIOProvider also passes Microsoft conformance tests and is compatible with ADO.

Exclusive (member level lock) and multiple user (record level lock) access are supported on a per-connection basis. The provider implements both immediate and delayed update modes supported by OLE DB. Immediate update mode maps directly to the SAS locking model; update mode is supported through an optimistic locking strategy implemented on top of the SAS locking strategy. These features allow concurrent access by multiple users to data sets on the RIO server.

Client scenarios for this provider are similar to those of BaseProvider, with the exception that RIOProvider's command and multi-user capabilities make it suitable for a wider range of tasks.

SAS/SHARE OLE DB SHARE PROVIDER

SAS/SHARE software provides a multi-user client/server environment for the SAS System. The server provides concurrent update access to SAS data for local and remote users across the enterprise. The server also

[1] At press time, pricing, beta schedules and target release date were not available.

provides remote users with low-overhead connectivity for reading SAS data.

ShareProvider connects OLE DB consumers to SAS/SHARE servers, including Version 6 installations. It allows the consumer to read and modify data sets residing on a Share server.

Availability

The SAS OLE DB ShareProvider will ship with an upcoming release of the SAS Share server as an optional component.[1]

Features

ShareProvider supports both read-only and update access to data files; it also allows the creation of new data sets. It does not have full SQL command capability, but data subsetting via a "where clause" feature is under consideration.

Locking and access modes are similar to the RIO provider, making it a viable multi-user solution. ShareProvider supports V6, V7 and V8 Share servers.

SAS/MDDB™ SERVER OLE DB FOR OLAP PROVIDER

SAS/MDDB Server is a specialized storage facility where data may be collected from a data warehouse or other data sources for storage in a matrix-like format for fast and easy access by tools such as multidimensional data viewers. This server is designed for high-performance access to large amounts of summarized data for complex multidimensional analysis and OLAP reporting.

The MDDB provider is quite different from the prior group of providers. Most obviously, it uses the OLE DB for OLAP extensions to provide multi-dimensional data.

Availability

The SAS/MDDB Server OLE DB for OLAP Provider will ship with upcoming releases of the SAS MDDB Server and CFO vision and as an optional component. It will be available for 6.12 maintenance releases as a beta product, and will ship as production software in version 8 based releases of the OLAP servers. At this time, there is no additional charge planned for the provider.

Features

This provider supports two of the three extended OLE DB for OLAP interfaces, and has full command support, including the MDX extensions for SQL. Due to the nature of the OLAP data it is serving, this will be a read only provider. The MDDB provider will support multi-user configurations. In addition, there will be support components for server catalog meta-data and server administration.

[1] At press time, pricing, beta schedules and target release date were not available.

Using this provider will allow users to import their OLAP results directly into report generation and analysis OLE DB for OLAP consumers like Excel 2000, Crystal Reports.

SAS SPD SERVER OLE DB PROVIDER

The SPD server is a multi-user data server for data retrieval in data warehousing and decision support applications. It provides a high performance data store of large SAS data sets. [SASP23A]

SPD plans to develop OLE DB and OLE DB for OLAP providers in the near future. No further details are available at this time.

CONCLUSION

OLE DB is an open, extensible standard that shrewdly leverages existing database access tools. It is slated to be a key technology in the next generation of Windows-based software.

It is no secret that the industry drive for open, collaborative software development is gathering momentum; driven by distributed, web-based technologies. The leaders in 21st century information delivery will be those who recognize this opportunity and seize it.

The goal of the SAS Nashville project is to put SAS Institute at the forefront of this trend, and the OLE DB providers and consumer are a critical component of that effort.

GLOSSARY OF TECHNICAL TERMS

ActiveX	A label used to refer to COM-based components and technologies, especially those related to the internet. (See also OLE)
ADO	Active Data Objects: an abstraction layer to provide OLE DB connectivity to
API	Application Programming Interface: a collection of functions or procedures that provide a specific software capability.
COM	The OLE Component Object Model, a specific approach to Object-Oriented software development developed by Microsoft.
Consumer	Any software component that utilizes an OLE DB interface. [OLEDB]
Driver	In ODBC, a software component providing access to a specific DBMS or file format.
Encapsulation	A protective programming technique that prevents external code from modifying the contents of data structures internal to an object or module.
Inheritance	A programming technique that facilitates the re-use of programming constructs. In COM, interface specifications are inherited rather than method implementation.
IOM	Interface Object Model: a SAS Nashville initiative to support distributed,

	component-based development.
ISV	Independent software vendor
Interface	An immutable collection of methods.
MDDB	Multi-Dimensional Database
Method	A function implemented by an object.
Object	In COM, a binary construct that provides a set of interfaces.
ODBC	Open Database Connectivity: a Microsoft API for accessing SQL-based data sources.
OLAP	On-Line Analytical Processing: commonly refers to the consolidated storage and multi-dimensional analysis of enterprise data.
OLE	Originally an acronym for *Object Linking and Embedding*, it is now used simply as a label to describe many COM based technologies. (See also ActiveX)
Polymorphism	A programming technique that allows disparate object types to be treated homogeneously. In COM, inheriting immutable interface specifications provides this.
Provider	Any software component that exposes an OLE DB interface. *Data providers* own or present data, while *service providers* apply an operation or transformation [OLEDB]
RDBMS	Relational Database Management System
RIO	Remote I/O: A SAS Nashville project initiative for distributed I/O.
SQL	Originally an acronym for Structured Query Language, now a label for the language used to retrieve specific data from an RDBMS

REFERENCES

[Brockschmidt] Brockschmidt, Kraig, *Inside OLE, Second Edition* (1995) Microsoft Press

[Chappell] Chappell, David. *Understanding ActiveX and OLE: A Guide for Developers & Managers* (1996), Microsoft Press
[SASC98] "SAS Institute's Nashville Project" (1998), SAS Communications, 4Q, 26-27

[Datamation] *Datamation: IS Managers Workbench* (1996) http://www.datamation.com/PlugIn/issues/1996/april15/04bevalgls.html

[Geiger] Geiger, Kyle, *Inside ODBC* (1995), Microsoft Press

[Melton] Melton, Jim, and Simon, Alan. *Understanding the New SQL: A Complete Guide* (1993), Morgan-Kauffman Publishers Inc.

[OLEDB] *Microsoft OLE DB 1.1 Programmer's Reference and Software Development Kit* (1997), Microsoft Press

[SASC98] "SAS Institute's Nashville Project" (1998), SAS Communications, 4Q, 26-27

[SASP23] Gona, Vino; Van Wyk, Jana "Version 7 Enhancements to SAS/ACCESS Software", *Proceedings of the Twenty-Third Annual SAS(R) Users Group International Conference - CD-ROM Version,* SAS Publications

[SASP23A] Advanced Server Development, "Scalable Performance Data Server™ – 2.0", *Proceedings of the Twenty-Third Annual SAS(R) Users Group International Conference - CD-ROM Version,* SAS Publications

ACKNOWLEDGMENTS

Forrest Boozer, SAS Institute Inc.
Brian Hess, SAS Institute Inc.
Randy Pierce, SAS Institute Inc.
David Shamlin, SAS Institute Inc.
Mitchell Hughes, Microsoft Corporation.

TRADEMARKS

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Thomas W. Cox
SAS Institute Inc.
SAS Campus Dr R/4281
Cary NC 27513
Work Phone: 919.677.8000 press 1,4875
Fax: 919.677.4444
Email:sasthc@wnt.sas.com