

Getting Started with SAS/ACCESS Interface to OLE DB

Introduction:

SAS/ACCESS Interface to OLE DB enables SAS to access a wide variety of data, such as database management system (DBMS) tables, email files, text files as well as OLAP data to name a few. This SAS/ACCESS interface accesses data from these sources through the OLE DB data providers. For those of you familiar with ODBC, an OLE DB provider is analogous to an ODBC driver, exposing data to an application. For more information about the OLE DB API, see the Microsoft OLE DB reference documentation

This paper is intended to be a gentle introduction to SAS/ACCESS Interface to OLE DB. It will discuss what software is needed for its use, how to use it (which includes syntax and examples), debugging and performance tips, and finally documentation and other resources.

Before you begin:

Before you begin using SAS/ACCESS Interface to OLE DB there are system requirements that will need to be met prior to its use. They include the following:

Products Required:

- ❖ Base SAS software
- ❖ SAS/ACCESS Interface to OLE DB

DBMS Products Required:

- ❖ An OLE DB provider needs to be installed on the same machine that SAS will be running on. The OLE DB provider is obtained from a third party (other than SAS) vendor.
- ❖ Any additional DBMS client software may need to be installed and configured on the same machine as the OLE DB provider. Many OLE DB providers communicate with the DBMS server using the DBMS client. Once the client software is installed and configured to communicate with the server in question, the OLE DB provider should be able to also communicate with the DBMS server.
- ❖ Microsoft Data Access Components (MDAC), Version 2.7 or higher

Note: The SAS System will automatically install Microsoft Data Access Components (MDAC) when you install SAS/ACCESS Interface to OLE DB software.

Once the above software has been installed and any additional client software has been installed and configured, SAS/ACCESS Interface to OLE DB is ready to be used.

Using SAS/ACCESS Interface to OLE DB

SAS/ACCESS Interface to OLE DB communicates with the data store via an underlying OLE DB provider using either the SAS/ACCESS LIBNAME statement or using the SQL Procedure Pass-Through Facility. Each of these methods will be briefly discussed below.

SAS/ACCESS LIBNAME Engine:

Using the SAS/ACCESS LIBNAME engine a SAS LIBRARY can be assigned that points directly to the (non-SAS) data that will be accessed. Once the library is assigned you can refer to the DBMS table using a two-level qualified table name as though it's a SAS data set, *Libref.tablename*.

The library can be assigned in one of two ways:

- ❖ using the LIBNAME statement, providing the OLEDB engine and appropriate connection options
LIBNAME *libref* SAS/ACCESS-engine-name SAS/ACCESS-engine-connection-options < SAS/ACCESS-LIBNAME options>;
- ❖ using the *New Library* window located on the SAS toolbar

The *New Library* window allows you to interactively assign the SAS library by providing all necessary information in each dialog. If desired, the option 'Enable at startup' can be selected to allow this library to be assigned automatically each time SAS is started.

Using the SAS/ACCESS LIBNAME Engine is referred to as 'implicit pass-through'. This is because when you write your SAS programs to access the DBMS tables using the library that was assigned, SAS will build the SQL native to the interface and pass as much as it can over to the database for processing. For example, the following SAS data step code written to access a Microsoft SQL Server table:

```
libname mydblib oledb init_string=" Provider=SQLOLEDB.1;Initial Catalog=users;Data Source=dwtsrv1 ";
data new;
set mydblib.emp;
run;
```

will actually be translated and be passed to the DBMS server as the following:

```
'select * from emp'
```

For specific examples of LIBNAME engine refer to the *EXAMPLES* section later in this document.

Proc SQL Pass-Through:

Proc SQL Pass-Through can be used to connect to a DBMS server or data source and send DBMS specific statements directly to the database server for execution. This facility is an alternative to the SAS/ACCESS LIBNAME statement.

The Pass-Through Facility enables you to:

- ❖ establish and terminate connections with a DBMS using the facility's [CONNECT Statement](#) and [DISCONNECT Statement](#)
- ❖ send dynamic, non-query, DBMS-specific SQL statements to a DBMS using the facility's [EXECUTE Statement](#)
- ❖ retrieve data directly from a DBMS using the facility's [CONNECTION TO Component](#) in the FROM clause of a PROC SQL SELECT statement.

The syntax for Proc SQL Pass-Through is as follows:

```
PROC SQL <options-list>;

CONNECT TO dbms-name <AS alias> <(<connect-statement-arguments> <database-connection-arguments>)>;

EXECUTE (DBMS-specific-SQL-statement) BY dbms-name | alias;

SELECT * FROM CONNECTION TO dbms-name | alias (DBMS-query) ;

DISCONNECT FROM <dbms-name | alias>;
```

Proc SQL Pass-Through facility is referred to as 'Explicit pass-through'. This is because the portion of the SQL contained in the parentheses is explicitly passed over 'as-is' to the database for processing. This will ensure that the SQL as seen in the SAS program is what is passed and processed on the server.

For specific Proc SQL Pass-Through examples refer to the *EXAMPLES* section later in this document.

Examples

Example 1: Libname statement as a prompted connection

This example uses a prompted connection to connect by prompting for all necessary information in each dialog. In this example the Microsoft Jet OLE DB provider is selected to access a Microsoft Access database. However, the method of using a prompted connection can be used for any OLE DB provider.

```
libname mylib oledb;
%put &sysdbmsg;          /* Note: For SAS V9 this would look like %put %superq(SYSDBMSG); */
```

Once the library above has been assigned, the connection information that was used is stored in the SAS macro variable, `&sysdbmsg`. The contents of the macro variable can be written to the SAS log using `%put`. The string in the SAS log would look something like the following:

```
OLEDB: Provider=Microsoft.Jet.OLEDB.4.0;Password="";Data Source=C:\My Documents\db2.mdb;
Persist Security Info=True
```

Using this information, the above prompted connection `libname` statement can be converted into a non-prompted connection by copying from the SAS log everything after the `'OLEDB:'` and pasting it as the `init_string` parameter within quotes as follows:

```
libname mylib oledb init_string="Provider=Microsoft.Jet.OLEDB.4.0;Password="";
Data Source=C:\My Documents\db2.mdb;Persist Security Info=True";
```

Now, the library, **MYLIB**, can be used to access the individual tables contained in the Microsoft Access database, `db2.mdb` in your SAS programs using data step, SAS procedures, etc. For example, the following example will access the Microsoft Access table, `table1`, and create a SAS data set `work.new` with variables `empid` and `lname` only.

```
data new (keep=empid lname);
set mylib.table1;
run;
```

Example 2: Libname statement connecting to Microsoft SQL Server (using NT Authentication)

```
libname x oledb provider=sqloledb dsn='trc-sql' properties=("Integrated Security"=SSPI "Persist Security Info"=True
"Initial Catalog"=Northwind);
```

NOTE: `Initial catalog`=database name, `dsn`= Microsoft SQL Server name

Now, using the libref of `'x'` you can create a Microsoft SQL Server table, `EMP`, from SAS data set `,work.mytable`, where `gender='M'` as follows:

```
proc sql;
create table x.EMP as select * from mytable where gender='M';
quit;
```

Example 3: Proc SQL Pass-Through connecting to Microsoft SQL Server (using SQL Server Authentication)

The following example connects to a Microsoft SQL Server using SQL Server Authentication. Note the connection parameters in this example. The `init_string` option is NOT used. The exact provider name and options required by the provider are explicitly coded. Because this example uses Proc SQL Pass-Through everything in the set of parentheses (*select * from employees where jobcode='602'*) is passed 'as-is' directly to the DBMS server.

```
proc sql;
connect to oledb(provider=SQLOLEDB dsn='dwtsrv1\dwt2000' user=dbitest password=dbigrp1);
create table blah as select * from connection to oledb(select * from employees where jobcode='602');
quit;
```

NOTE: `dsn`= Microsoft SQL Server name

Example 4: Libname statement connecting to Excel using the Microsoft Jet OLE DB Provider

This example assigns a libref, **mylib**, to the Excel file, `emp.xls`. Using this libref, the Excel worksheet, `Sheet1`, is read. Note the `'$'` is used after the worksheet name to designate this as a worksheet and not a range.

```
libname mylib oledb provider=JET provider_string='Excel 8.0' datasource='c:\emp.xls';
```

```
data new;
set mylib.'Sheet1'$n;
run;
```

**Example 5: Using SCHEMA libname option (useful when the library assigns, but is empty!)
Libname using OLE DB provider for ODBC drivers**

```
libname ora oledb Provider=msdasql dsn=oracle pwd=tiger uid=scott;
```

NOTE: **dsn**=name of the ODBC Data source name

If opening the library, ORA, shows no tables more than likely the use of the SCHEMA libname option is needed. By default, the OLE DB libname engine will use the userid supplied for connecting, as the schema when getting a list of tables. If your database has been set to use a schema other than your userid then you will need to supply the schema= option on the libname statement. For example, if your database has been set up so that you have tables listed under the schema of marketing and your userid allows you to manipulate those tables, your libname statement would be:

```
libname ora oledb Provider=msdasql dsn=oracle pwd=tiger uid=scott userid schema=marketing;
```

If the following data step was submitted to find out all the marketing contacts for the state of NC:

```
data work.contacts;
set ora.contacts;
where state="NC";
run;
```

The statement that is passed to the database is:

```
SELECT "name", "state" FROM "marketing"."CONTACTS" WHERE ("state" = 'NC' )
```

Example 6: Libname connection to a Microsoft Access database that requires a workgroup file

Accessing a Microsoft Access database that has security setup such that a workgroup file is used, the workgroup file has to be passed to the Microsoft Jet OLEDB provider via a specific Jet property which is done via the *init_string* property.

```
libname x oledb init_string='Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\testing\secure.mdb;Persist Security Info=False;
Jet OLEDB:System database=c:\testing\system1.mdw;Jet OLEDB:Database Password=dbitest';
```

NOTE: **Jet OLEDB:System database** = the name and location of their work group information file,
Jet OLEDB:Database Password = the password of the database if one exists

Example 7: Calling a DBMS stored procedure

SAS/ACCESS to OLEDB can be used to call a DBMS stored procedure. The first example below calls the Microsoft SQL Server stored procedure 'sp_who'. Because this stored procedure returns a result set the procedure name must be specified in the *CONNECTION TO* component of the from clause using Proc SQL Pass-Through.

```
proc sql;
connect to oledb(dsn='dwtsrv1' uid=dbitest pwd=dbigrp1 provider=sqloledb);
select * from connection to oledb (sp_who);
quit;
```

When calling a stored procedure that does NOT return a result set it can be executed via the execute() statement as the following example shows:

```
proc sql;
connect to oledb(dsn=dwtsrv1 uid=dbitest pwd=dbigrp1 provider=sqloledb);
execute (sp_rename "BADCRIME", "BARBCRIME") by oledb;
quit;
```

Example 8: Accessing OLAP data

The SAS/ACCESS interface to OLE DB provides a facility for accessing OLE DB for OLAP data. You can specify a Multidimensional Expressions (MDX) statement through the Pass-Through Facility to access the data directly. This implementation provides read-only access to OLE DB for OLAP data. The following example accesses a Microsoft SQL Server cube.

```
PROC SQL;
CONNECT TO OLEDB (PROVIDER=MSOLAP PROPS=('INITIAL CATALOG'= 'FoodMart 2000'
'DATA SOURCE'= 'dwtsrv1'));
SELECT * FROM CONNECTION TO OLEDB
(MDX::Select NON EMPTY { [Time].[1997], [Time].[1998] } ON COLUMNS, NON
EMPTY { [Account].[All Account] } ON ROWS From [Budget] Where ([Category].
[All Category],[Measures].[Amount],[Store].[All Stores] ) );
QUIT;
```

More examples of accessing OLAP data can be found on the PC at *!sasroot\access\sample\mdxsamp.sas*. For more information about MDX syntax refer to the Microsoft Data Access Components Software Developer's Kit.

Debugging tips

You now have your programs written to access the data using OLE DB, but things are not behaving quite as you expected. Maybe you are not seeing the tables you are expecting or perhaps the query is running very slowly. Below are some tips that can be used in SAS to help solve these sorts of problems.

Sastrace:

As discussed above, SAS/ACCESS Interface to OLE DB offers LIBNAME engine as one means for connecting to the data source. When using the Libname engine, SAS builds the DBMS specific SQL and tries to push as much of the query over to the DBMS server. In order to see the DBMS specific SQL that's being passed, the following statement can be added to the SAS program.

```
options sastrace=',,d' sastraceloc=saslog nostsuffix;
```

To turn off sastrace the following can be submitted:

```
options sastrace=none;
```

If the entire query is NOT shown in the sastrace portion of the SAS log, chances are for some reason SAS could not pass the entire query to the DBMS server and is doing most or all of the processing. For reasons behind this see the *Performance Tips* section below.

Special Queries:

Many databases provide the use of system tables or support special queries that can be used to find out information such as list of tables, columns, or other useful information. SAS/ACCESS Interface to OLEDB offers special queries that allow this sort of information to be returned to SAS. These special queries can only be accessed using the Proc SQL Pass-Through facility. The full list of the supported queries can be found in the SAS documentation. This paper will only highlight the most often used.

❖ **OLEDB::Tables:** Returns a list of all tables available to a given user via the OLE DB provider

```
proc sql;
connect to oledb(init_string="Provider=.....");
create table MyTables as select * from connection to oledb(OLEDB::Tables);
quit;
```

In addition to the table names, OLEDB::Tables also returns the SCHEMA associated with each table. Because the SAS

libname engine shows tables in the current userid's schema very often people find the SAS library to be empty because we are not looking in the correct schema. So, using `OLEDB::Tables` one can get the correct schema name and add it to the SAS library using the `SCHEMA` libname option.

```
LIBNAME mylib OLEDB ...SCHEMA=dbo;
```

- ❖ **OLEDB::Columns:** Returns a list of column information (such a data type, field length, etc) in a specified table via the OLE DB provider.

The following example will return column information for the table, *myTable*.

```
proc sql;
connect to oledb(init_string="Provider=..... ");
create table MyColumns as select * from connection to oledb(OLEDB::Columns, "myTable");
quit;
```

Performance Tips

SAS/ACCESS Interface to OLE DB does offer options that can be used to improve performance. The following list includes some of these options as well as other things that can be used to improve performance:

- ❖ **Buffering reads:** The `READBUFF` option can be used to specify the number of rows of DBMS data to read into the buffer. This option can be set as a libname option, data set option, or in the connect statement of Proc SQL Pass-Through. The default value for `READBUFF` is 1. Increasing this value can improve read performance, however, setting it too high can actually degrade performance as more memory will be used.

```
libname mylib oledb ...READBUFF=1000;
```

- ❖ **Ensuring the DBMS server does all the work:** Whether using the Libname engine or Proc SQL Pass-Through the goal of an efficient query is to pass the query over to the DBMS server to do all the processing and return only the result set to SAS. When using Proc SQL Pass-Through, as discussed above, the query specified is passed to the DBMS server. However, when using the libname engine there are certain circumstances where SAS can not convert the SAS-specific code into DBMS-specific SQL. One of the most common causes is the use of a SAS function in a where clause that cannot be converted to a DBMS equivalent function. However, using SAS 9.0 and above, using the `SQL_FUNCTIONS=ALL` libname option allows the engine to pass more functions to the DBMS server. There are other situations that may cause joins to not be passed. These are discussed in further detail in the SAS Online documentation as well as in the following SUGI Paper:

'Using the SAS/ACCESS Libname Technology to Get Improvements in Performance and Optimizations in SAS/SQL Queries:'
<http://www2.sas.com/proceedings/sugi26/p110-26.pdf>

Keep in mind that if you suspect your query is not being passed in its entirety adding the `sastrace` option to your program, as discussed in the *debugging* section above, can be used to display the SQL SAS is passing in the SAS log.

- ❖ **Retrieving only the rows and columns necessary:** Limiting the amount of data returned to SAS will improve the performance. This can be done by only selecting columns needed and by limiting the number of rows that the DBMS server returns to SAS by subsetting with a `WHERE` clause.
- ❖ **Bulkloading:** Using the libname option, `BULKLOAD=YES`, enables you to efficiently insert rows of data into a Microsoft SQL Server table as a unit. This option is only available with the Microsoft SQL Server OLE DB provider.
- ❖ **Insertbuff:** Beginning in SAS 9, the libname or dataset option, `INSERTBUFF`, can be used to improve the performance of writing data to the DBMS server. The `INSERTBUFF` option can be used to specify the number of rows to insert. The optimal value for this option varies with factors such as network type and available memory. Experimenting with different values in order to determine the best value for the particular configuration is recommended.

Documentation/Reference/SUGI Papers For further information the following documentation is also available:

- ❖ SAS/ACCESS Software for Relational Databases: Reference (available in the SAS online doc CD or SAS help)
- ❖ What's up with OLE DB: <http://www2.sas.com/proceedings/sugi24/Dataware/p136-24.pdf>
- ❖ Comparing OLE DB and ODBC: <http://ftp.sas.com/techsup/download/v8papers/odbcdb.pdf>
- ❖ Using the SAS/ACCESS Libname Technology to Get Improvements in Performance and Optimizations in SAS/SQL Queries: <http://www2.sas.com/proceedings/sugi26/p110-26.pdf>
- ❖ Accessing Microsoft EXCEL and Microsoft Access Through the Use of a Simple Libname Statement: <http://www2.sas.com/proceedings/sugi27/p025-27.pdf>
- ❖ Microsoft MSDN Library: Doc on ODBC, OLE DB, etc: <http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001860>