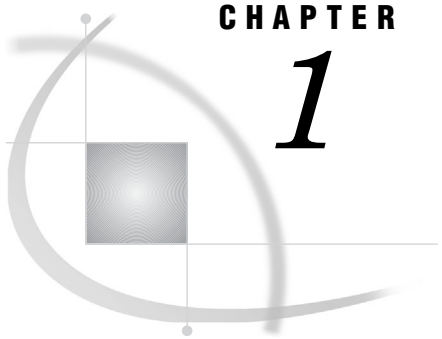

CHAPTER

1

SAS/ACCESS for OLE DB

Introduction to the SAS/ACCESS Interface to OLE DB

This document includes details *only* about the SAS/ACCESS interface to OLE DB. It should be used as a supplement to the generic SAS/ACCESS documentation *SAS/ACCESS for Relational Databases: Reference*.

Microsoft OLE DB is an API (application programming interface) that provides access to data that can be in a database table, an e-mail file, a text file, or another type of file. This SAS/ACCESS interface accesses data from these sources through OLE DB data providers such as Microsoft Access, Microsoft SQL Server, and Oracle.

LIBNAME Statement Specifics for OLE DB

This section describes the LIBNAME statement as supported in the SAS/ACCESS interface to OLE DB. For a complete description of this feature, see the LIBNAME statement section in *SAS/ACCESS for Relational Databases: Reference*. The OLE DB specific syntax for the LIBNAME statement is as follows:

LIBNAME *libref* **oledb** <*connection-options*> <*LIBNAME-options*>;

Arguments

libref
is any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

oledb
is the SAS/ACCESS engine name for the interface to OLE DB.

connection-options
provide connection information and control how SAS manages the timing and concurrence of the connection to the data source. You can connect to a data source either by using OLE DB Services or by connecting directly to the provider. For details about each of these methods, see "Connecting with OLE DB Services" on page 6 and "Connecting Directly to a Data Provider" on page 7.

The following connection options are available with both connection methods:

USER=<'>*user-name*<'>
enables you to connect to an OLE DB data source with a user ID that is different from the default ID. The default is your user ID.

PASSWORD=<'>*password*<'>
specifies the OLE DB password that is associated with your user ID.

Note: If you do not wish to enter your OLE DB password in uncoded text, see PROC PWENCODE for a method to encode it. △

DATASOURCE=<'>*data-source*<'>
identifies the data source object (such as a relational database server or a local file) to which you want to connect.

PROVIDER=<'>*provider-name*<'>
specifies which OLE DB provider to use to connect to the data source. This option is required during batch processing.

There is no restriction on the length of the *provider-name*. If the *provider-name* contains blank spaces or special characters, enclose it in quotation marks.

If you do not specify a provider, an OLE DB Services dialog box prompts you for connection information. In batch mode, if you do not specify a provider the connection fails.

If you are using the Microsoft Jet OLE DB 4.0 provider, specify PROVIDER=JET.

PROPERTIES=(<'>*property-1*<'>=<'>*value-1*<'> < . . .
<'>*property-n*<'>=<'>*value-n*<'>>)
specifies standard provider properties that enable you to connect to a data source and to define connection attributes. If a property name or value contains embedded spaces or special characters, enclose the name or value in quotation marks. Use a blank space to separate multiple properties.

Note: See your provider's documentation for a list and description of all the properties that your provider supports. △

No properties are specified by default.

PROVIDER_STRING=<'>*extended-properties*<'>
specifies provider-specific extended connection information, such as the file type of the data source. If the string contains blank spaces or special characters, enclose it in quotation marks. For example, the Microsoft Jet provider accepts strings that indicate file type (such as 'Excel 8.0').

The following example uses the Jet 4.0 provider to access the spreadsheet Y2KBUDGET.XLS. Specify the 'Excel 8.0' provider string so that Jet recognizes the file as an Excel 8.0 worksheet.

```
libname budget oledb provider=jet provider_string='Excel 8.0'
                     datasource='d:\excel80\Y2Kbudget.xls';
```

OLEDB_SERVICES=YES | NO
determines whether SAS uses OLE DB Services to connect to the data source. Specify YES to use OLE DB Services or specify NO to use the provider to connect to the data source.

Specifying PROMPT=YES and OLEDB_SERVICES=YES enables you to set more options than you would otherwise be able to set (by being prompted by the provider's dialog box). If OLEDB_SERVICES=NO, you must specify PROVIDER= first in order for the provider's prompt dialog boxes to be used. If PROVIDER= is omitted, SAS uses OLE DB Services, even if you specify OLEDB_SERVICES=NO.

YES is the default.

For Microsoft SQL Server data, if BULKLOAD=YES, then OLEDB_SERVICES= is set to NO.

When OLEDB_SERVICES=YES and a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable.

PROMPT =YES | NO
determines whether an interactive dialog box is displayed to guide you through the connection process. The interactive dialog box is one of the following:

- an OLE DB provider dialog box if OLEDB_SERVICES=NO and you specify a provider.
- an OLE DB Services dialog box if OLEDB_SERVICES=YES or if you do not specify a provider.

The OLE DB Services dialog box is generally preferred over the provider's dialog box because the OLE DB Services dialog box enables you to set options more easily.

If you specify a provider and set OLEDB_SERVICES=NO, the default is PROMPT=NO. Otherwise, the default is PROMPT=YES.

If OLEDB_SERVICES=YES or if you do not specify a provider, an OLE DB Services dialog box displays even if you specify PROMPT=NO.

Specify no more than one of the following options on each LIBNAME statement: COMPLETE=, REQUIRED=, PROMPT=.

Any properties that you specify in the PROPERTIES= option are displayed in the prompting interface, and you can edit any field.

UDL_FILE=<'>*path-and-file-name*<'>
specifies the path and filename for a Microsoft universal data link (UDL). For example, you could specify
UDL_FILE="C:\WinNT\profiles\me\desktop\MyDBLink.UDL"

This option does not support SAS filerefs. SYSDBMSG is *not* set on successful completion. For more information, see Microsoft's documentation about the Data Link API.

This option overrides any values that are set with the INIT_STRING=, PROVIDER=, and PROPERTIES= options.

The following connection option is only available when you use OLE DB Services:

INIT_STRING='*property-1=value-1<...;property-n=value-n>*'
specifies an initialization string, enabling you to bypass the interactive prompting interface yet still use OLE DB Services. Use a semicolon to separate properties.

After you connect to a data source, SAS returns the complete initialization string to the macro variable SYSDBMSG, which stores the connection information that you specify in the prompting window. You can reuse the initialization string to make automated connections or to specify connection information for batch jobs.

For example, if you specify the following initialization string:

```
init_string='Provider=SQLOLEDB;Password=dbmgr1;Persist
Security Info=True;User ID=rachel;Initial Catalog=users;
Data Source=dwtsrv1';
```

then the content of the SYSDBMSG macro variable is:

```
OLEDB:  Provider=SQLOLEDB;Password=dbmgr1;
Persist Security Info=True;User ID=rachel;
Initial Catalog=users;Data Source=dwtsrv1;
```

If you store this string for later use, delete the **OLEDB:** prefix and any initial spaces before the first listed option.

There is no default value. However, if you specify a null value for this option, the OLE DB Provider for ODBC (MSDASQL) is used with your default data source and its properties. See your OLE DB documentation for more information about these default values.

This option overrides any values that are set with the PROVIDER= and PROPERTIES= options.

To write the initialization string to the SAS log, submit the following code immediately after connecting to the data source: **%put %superq(SYSDBMSG);**

This option is not available if OLEDB_SERVICES=NO.

The following connection options are only available when you connect directly to a provider:

COMPLETE=YES | NO
specifies whether SAS attempts to connect to the data source without prompting you for connection information.

If you specify COMPLETE=YES and the connection information that you specify in your LIBNAME statement is sufficient, then SAS makes the connection and does not prompt you for additional information.

If you specify COMPLETE=YES and the connection information that you specify in your LIBNAME statement is not sufficient, the provider's dialog box prompts you for additional information. You can enter optional information as well as required information in the dialog box.

NO is the default value.

COMPLETE= is available only when you set OLEDB_SERVICES=NO and you specify a provider. It is not available in the Pass-Through Facility.

Specify no more than one of the following options on each LIBNAME statement: COMPLETE=, REQUIRED=, PROMPT=.

REQUIRED=YES | NO
specifies whether SAS attempts to connect to the data source without prompting you for connection information and whether you can interactively specify optional connection information.

If you specify REQUIRED=YES and the connection information that you specify in your LIBNAME statement is sufficient, SAS makes the connection and you are not prompted for additional information.

If you specify REQUIRED=YES and the connection information that you specify in your LIBNAME statement is not sufficient, the provider's dialog box prompts you for the required connection information. You cannot enter optional connection information in the dialog box.

NO is the default value.

REQUIRED= is available only when you set OLEDB_SERVICES=NO and you specify a provider in the PROVIDER= option. It is not available in the Pass-Through Facility

Specify no more than one of the following options on each LIBNAME statement: COMPLETE=, REQUIRED=, PROMPT=.

LIBNAME-options
define how DBMS objects are processed by SAS. Some LIBNAME options can enhance performance; others determine locking or naming behavior. The following table describes the LIBNAME options that are suupported for OLE DB, and presents default values where applicable. See the section about the SAS/ACCESS LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for detailed information about these options.

Table 1.1 SAS/ACCESS LIBNAME Options for OLE DB

Option	Default Value
ACCESS=	none
AUTOCOMMIT=	data source specific
BL_KEEPIDENTITY=	NO
BL_KEEPNULLS=	YES
BL_OPTIONS=	not specified
BULKLOAD=	NO
CELLPROP=	VALUE
COMMAND_TIMEOUT=	0 (no timeout)
CONNECTION=	SHAREDREAD
CONNECTION_GROUP=	none
CURSOR_TYPE=	none
DBCOMMIT=	1000 (inserting) or 0 (updating)
DBCONINIT=	none
DBCONTERM=	none
DBCREATE_TABLE_OPTS=	none
DBGEN_NAME=	DBMS
DBINDEX=	NO
DBLIBINIT=	none

Option	Default Value
DBLIBTERM=	none
DBMAX_TEXT=	1024
DBNULLKEYS=	YES
DEFER=	NO
DELETE_MULT_ROWS=	NO
DIRECT_EXE=	none
DIRECT_SQL=	YES
IGNORE_ READ_ONLY_COLUMNS=	NO
INSERT_SQL=	data source specific
INSERTBUFF=	1
MULTI_DATASRC_OPT=	NONE
PRESERVE_COL_NAMES=	see “Naming Conventions for OLE DB” on page 22
PRESERVE_TAB_NAMES=	see “Naming Conventions for OLE DB” on page 22
QUALIFIER=	none
QUALIFY_ROWS=	NO
QUOTE_CHAR=	not set
READBUFF=	1
READ_LOCK_TYPE=	see “Locking in the OLE DB Interface” on page 18
READ_ISOLATION_LEVEL=	not set (see “Locking in the OLE DB Interface” on page 18)
REREAD_EXPOSURE=	NO
SCHEMA=	none
SPOOL=	YES
SQL_FUNCTIONS=	NONE
STRINGDATES=	NO
UPDATE_ISOLATION_LEVEL=	not set (see “Locking in the OLE DB Interface” on page 18)
UPDATE_LOCK TYPE=	ROW
UPDATE_MULT_ROWS=	NO
UTILCONN_TRANSIENT=	NO

Connecting with OLE DB Services

By default, the SAS/ACCESS interface to OLE DB uses OLE DB Services because this is often the fastest and easiest way to connect to a data provider.

OLE DB Services provides performance optimizations and scaling features, including resource pooling. It also provides interactive prompting for the provider name and connection information.

When you submit a simple LIBNAME statement such as

```
libname mydblib oledb;
```

SAS directs OLE DB Services to display a dialog box that contains tabs where you can enter the provider name and connection information.

Note: After you make a successful connection using OLE DB Services, you can retrieve the connection information and reuse it in batch jobs and automated connections. For more information, see the connection options INIT_STRING= and OLEDB_SERVICES=. △

Connecting Directly to a Data Provider

To connect to a data source, the SAS/ACCESS interface to OLE DB requires a provider name and provider-specific connection information such as the user ID, password, schema, or server name. If you know all of this information, you can connect directly to a provider, without using OLE DB Services.

Note: If you are connecting to Microsoft SQL Server and specifying the SAS/ACCESS option BULKLOAD=YES, then you must connect directly to the provider. △

To connect directly to a provider, you must specify the following:

- the name of the provider (PROVIDER=)
- that you do not want to use OLE DB Services (OLEDB_SERVICES=NO)
- any required connection information.

After you connect to your provider, you can use a special OLE DB query called PROVIDER_INFO to make subsequent non-prompted connections easier. You can submit this special query as part of a PROC SQL query in order to display all of the available provider names and properties. For an example, see “Examples of Special OLE DB Queries” on page 13.

If you know only the provider name and you are running an interactive SAS session, you can be prompted for the provider’s properties. Specify PROMPT=YES to direct the provider to prompt you for properties and other connection information. Each provider displays its own prompting interface.

If you run SAS in a batch environment, specify only USER=, PASSWORD=, DATASOURCE=, PROVIDER=, PROPERTIES=, and OLEDB_SERVICES=NO.

OLE DB LIBNAME Statement Examples

In the following example, the libref MYDBLIB uses the SAS/ACCESS OLE DB engine to connect to a Microsoft SQL Server database.

```
libname mydblib oledb user=username password=password datasource=dept203
                   provider=sqloledb properties=('initial catalog'=mgronly);

proc print data=mydblib.customers;
   where state='CA';
run;
```

In the following example, the libref MYDBLIB uses the SAS/ACCESS engine for OLE DB to connect to an Oracle database. Because prompting is enabled, you can review and edit the user, password, and data source information in a dialog box.

```
libname mydblib oledb user=username password=password datasource=v2o7223.world
                      provider=msdaora prompt=yes;
```

```
proc print data=mydblib.customers;
   where state='CA';
run;
```

In the following example, you submit a basic LIBNAME statement, so an OLE DB Services dialog box prompts you for the provider name and properties' values.

```
libname mydblib oledb;
```

The advantage of being prompted is that you do not need to know any special syntax to set the values for the properties. Prompting also enables you to set more options than you might when you connect directly to the provider (and do not use OLE DB Services).

Data Set Options for OLE DB

The following table describes the data set options that are supported for OLE DB, and provides default values where applicable. See the section about data set options in *SAS/ACCESS for Relational Databases: Reference* for detailed information about these options.

Table 1.2 SAS/ACCESS Data Set Options for OLE DB

Option	Default Value
BL_KEEPIDENTITY=	LIBNAME option setting
BL_KEEPNULLS=	LIBNAME option setting
BL_OPTIONS=	LIBNAME option setting
COMMAND_TIMEOUT=	LIBNAME option setting
CURSOR_TYPE=	LIBNAME option setting
DBCOMMIT=	LIBNAME option setting
DBCONDITION=	none
DBCREATE_TABLE_OPTS=	LIBNAME option setting
DBFORCE=	NO
DBGEN_NAME=	DBMS
DBINDEX=	LIBNAME option setting
DBKEY=	none
DBLABEL=	NO
DBMASTER=	none
DBMAX_TEXT=	1024
DBNULL=	_ALL_=YES
DBNULLKEYS=	LIBNAME option setting
DBSASTYPE=	see "Data Types for OLE DB" on page 22
DBTYPE=	see "Data Types for OLE DB" on page 22
ERRLIMIT=	1
IGNORE_ READ_ONLY_COLUMNS=	NO
INSERT_SQL=	LIBNAME option setting

Option	Default Value
INSERTBUFF=	LIBNAME option setting
NULLCHAR=	SAS
NULLCHARVAL=	a blank character
PRESERVE_COL_NAMES=	LIBNAME option setting
QUALIFIER=	LIBNAME option setting
READBUFF=	LIBNAME option setting
READ_ISOLATION_LEVEL=	LIBNAME option setting
SASDATEFMT=	not set
SCHEMA=	LIBNAME option setting
UPDATE_ISOLATION_LEVEL=	LIBNAME option setting
UPDATE_LOCK_TYPE=	LIBNAME option setting
UTILCONN_TRANSIENT=	YES

Pass-Through Facility Specifics for OLE DB

See the section about the Pass-Through Facility in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

The Pass-Through Facility specifics for OLE DB are as follows:

- The *dbms-name* is **OLEDB**.
- The CONNECT statement is required.
- PROC SQL supports multiple connections to OLE DB. If you use multiple simultaneous connections, you must use an *alias* to identify the different connections. If you do not specify an alias, the default alias, **OLEDB**, is used. The functionality of multiple connections to the same OLE DB provider might be limited by a particular provider.
- The CONNECT statement *database-connection-arguments* are identical to the LIBNAME connection options. For some data sources, the connection options have default values and are therefore not required.

  Note: Not all of the connection options are supported by all OLE DB providers. Refer to your provider documentation for more information. △
- The following LIBNAME options are available with the CONNECT statement:

  AUTOCOMMIT=
  CELLPROP=
  COMMAND_TIMEOUT=
  CURSOR_TYPE=
  DBMAX_TEXT=
  QUALIFY_ROWS=
  READ_ISOLATION_LEVEL=
  READ_LOCK_TYPE=
  READBUFF=
  STRINGDATES=.

Examples

The following example uses an alias to connect to a Microsoft SQL Server database and select a subset of data from the PAYROLL table. The SAS/ACCESS engine uses OLE DB Services to connect to OLE DB because this is the default action when the OLEDB_SERVICES= option is omitted.

```
proc sql;
connect to oledb as finance
  (user=username password=password datasource=dwtsrv1
   provider=sqloledb);

select * from connection to finance (select * from payroll
                                     where jobcode='FA3');
quit;
```

In the following example, the CONNECT statement omits the provider name and properties. An OLE DB Services dialog box will prompt you for the connection information.

```
proc sql;
connect to oledb;
quit;
```

The following example uses OLE DB Services to connect to a provider that is configured under the data source name **User's Data** with the alias USER1. Note that the data source name can contain quotation marks and spaces.

```
proc sql;
   connect to oledb as user1
   (provider=JET datasource='c:\db1.mdb');;
```

Special OLE DB Queries

The following special queries are supported by the SAS/ACCESS interface to OLE DB. Many databases provide or use system tables that enable queries to return the list of available tables, columns, procedures, and other useful information. In OLE DB, much of this functionality is provided through special APIs (application programming interfaces) in order to accommodate databases that do not follow the SQL table structure. You can use these special queries on non-SQL and on SQL databases.

Note: Not all of the queries are supported by all OLE DB providers. Refer to your provider documentation for more information. △

The general format of the special queries is as follows:

OLEDB::*schema-rowset*("*parameter 1*","*parameter n*")

where

OLEDB::
is required to distinguish special queries from regular queries.

schema-rowset
is the specific schema rowset that is being called. All valid schema rowsets are listed under the IDBSchemaRowset Interface in the *Microsoft OLE DB Programmer's Reference*. Both OLEDB:: and *schema-rowset* are case-sensitive.

"*parameter n*"
is a quoted string that is enclosed by commas. The values for the special query arguments are specific to each data source. For example, you supply the fully qualified table name for a "*Qualifier*" argument. In dBase, the value of "*Qualifier*" might be **c:\dbase\tst.dbf**, and in SQL Server, the value might be **test.customer**. In addition, depending on the data source that you use, values for an "*Owner*" argument might be a user ID, a database name, or a library. All arguments are optional. If you specify some but not all the arguments within a parameter, use commas to indicate the omitted arguments. If you do not specify any parameters, commas are not necessary. Note that these special queries might not be available for all OLE DB providers.

The following special queries are supported:

OLEDB::ASSERTIONS(<"*Catalog*", "*Schema*", "*Constraint-Name*">)
returns assertions defined in the catalog that are owned by a given user.

OLEDB::CATALOGS(<"*Catalog*">)
returns physical attributes associated with catalogs that are accessible from the DBMS.

OLEDB::CHARACTER_SETS(<"*Catalog*", "*Schema*","*Character-Set-Name*">)
returns the character sets defined in the catalog that are accessible to a given user.

OLEDB::CHECK_CONSTRAINTS(<"*Catalog*", "*Schema*", "*Constraint-Name*">)
returns check constraints defined in the catalog that are owned by a given user.

OLEDB::COLLATIONS(<"*Catalog*", "*Schema*", "*Collation-Name*">)
returns the character collations defined in the catalog that are accessible to a given user.

OLEDB::COLUMN_DOMAIN_USAGE(<"*Catalog*", "*Schema*", "*Domain-Name*", "*Column-Name*">)
returns the columns defined in the catalog that are dependent on a domain defined in the catalog and owned by a given user.

OLEDB::COLUMN_PRIVILEGES(<"*Catalog*", "*Schema*", "*Table-Name*", "*Column-Name*", "*Grantor*", "*Grantee*">)
returns the privileges on columns of tables defined in the catalog that are available to or granted by a given user.

OLEDB::COLUMNS(<"*Catalog*", "*Schema*", "*Table-Name*", "*Column-Name*">)
returns the columns of tables defined in the catalogs that are accessible to a given user.

OLEDB::CONSTRAINT_COLUMN_USAGE(<"*Catalog*", "*Schema*", "*Table-Name*", "*Column-Name*">)
returns the columns used by referential constraints, unique constraints, check constraints, and assertions that are defined in the catalog and owned by a given user.

OLEDB::CONSTRAINT_TABLE_USAGE(<"*Catalog*", "*Schema*", "*Table-Name*">)
returns the tables used by referential constraints, unique constraints, check constraints, and assertions that are defined in the catalog and owned by a given user.

OLEDB::FOREIGN_KEYS(<"*Primary-Key-Catalog*", "*Primary-Key-Schema*", "*Primary-Key-Table-Name*", "*Foreign-Key-Catalog*", "*Foreign-Key-Schema*", "*Foreign-Key-Table-Name*">)
returns the foreign key columns defined in the catalog by a given user.

OLEDB::INDEXES(<"*Catalog*", "*Schema*", "*Index-Name*", "*Type*", "*Table-Name*">)
returns the indexes defined in the catalog that are owned by a given user.

OLEDB::KEY_COLUMN_USAGE(<"*Constraint-Catalog*", "*Constraint-Schema*", "*Constraint-Name*", "*Table-Catalog*", "*Table-Schema*", "*Table-Name*", "*Column-Name*">)
returns the columns defined in the catalog that are constrained as keys by a given user.

OLEDB::PRIMARY_KEYS(<"*Catalog*", "*Schema*", "*Table-Name*">)
returns the primary key columns defined in the catalog by a given user.

OLEDB::PROCEDURE_COLUMNS(<"*Catalog*", "*Schema*", "*Procedure-Name*", "*Column-Name*">)
returns information about the columns of rowsets returned by procedures.

OLEDB::PROCEDURE_PARAMETERS(<"*Catalog*", "*Schema*", "*Procedure-Name*", "*Parameter-Name*">)
returns information about the parameters and return codes of the procedures.

OLEDB::PROCEDURES(<"*Catalog*", "*Schema*", "*Procedure-Name*", "*Procedure-Type*">)
returns procedures defined in the catalog that are owned by a given user.

OLEDB::PROVIDER_INFO()
returns output that contains the following columns: PROVIDER_NAME, PROVIDER_DESCRIPTION, and PROVIDER_PROPERTIES. The PROVIDER_PROPERTIES column contains a list of all the properties that the provider supports. The properties are separated by a semicolon(;). See the "Examples of Special OLE DB Queries" on page 13.

OLEDB::PROVIDER_TYPES(<"*Data Type*", "*Best-Match*">)
returns information about the base data types supported by the data provider.

OLEDB::REFERENTIAL_CONSTRAINTS(<"*Catalog*", "*Schema*", "*Constraint-Name*">)
returns the referential constraints defined in the catalog that are owned by a given user.

OLEDB::SCHEMATA(<"*Catalog*", "*Schema*", "*Owner*">)
returns the schemas that are owned by a given user.

OLEDB::SQL_LANGUAGES()
returns the conformance levels, options and dialects supported by the SQL implementation processing data that is defined in the catalog.

OLEDB::STATISTICS(<"*Catalog*", "*Schema*", "*Table-Name*">)
returns the statistics defined in the catalog that are owned by a given user.

OLEDB::TABLE_CONSTRAINTS(<"*Constraint-Catalog*", "*Constraint-Schema*", "*Constraint-Name*", "*Table-Catalog*", "*Table-Schema*", "*Table-Name*", "*Constraint-Type*">)
returns the table constraints defined in the catalog that are owned by a given user.

OLEDB::TABLE_PRIVILEGES(<"*Catalog*", "*Schema*", "*Table-Name*", "*Grantor*", "*Grantee*">)
returns the privileges on tables defined in the catalog that are available to or granted by a given user.

OLEDB::TABLES(<"*Catalog*", "*Schema*", "*Table-Name*", "*Table-Type*">)
returns the tables defined in the catalog that are available to or granted by a given user.

OLEDB::TRANSLATIONS(<"*Catalog*", "*Schema*", "*Translation-Name*">)
returns the character translations defined in the catalog that are accessible to a given user.

OLEDB::USAGE_PRIVILEGES(<"*Catalog*", "*Schema*", "*Object-Name*", "*Object-Type*", "*Grantor*", "*Grantee*">)
returns the USAGE privileges on objects defined in the catalog that are available to or granted by a given user.

OLEDB::VIEW_COLUMN_USAGE(<"*Catalog*", "*Schema*", "*View-Name*">)
returns the columns on which viewed tables, defined in the catalog and owned by a given user, are dependent.

OLEDB::VIEW_TABLE_USAGE(<"*Catalog*", "*Schema*", "*View-Name*">)
returns the tables on which viewed tables, defined in the catalog and owned by a given user, are dependent.

OLEDB::VIEWS(<"*Catalog*", "*Schema*", "*Table-Name*">)
returns the viewed tables defined in the catalog that are accessible to a given user.

For a complete description of each rowset and the columns that are defined in each rowset, refer to the Microsoft OLE DB Programmer's Reference.

Examples of Special OLE DB Queries

The following example retrieves a rowset that displays all of the tables that are accessed by the schema HRDEPT:

```
proc sql;
  connect to oledb(provider=sqloledb properties=("User ID"=testuser
                   Password=testpass
     "Data Source"='dwtsrv1'));
  select * from connection to oledb
       (OLEDB::TABLES(,"HRDEPT"));
quit;
```

The following example uses the special query OLEDB::PROVIDER_INFO() to produce the output that follows:

```
proc sql;
  connect to oledb(provider=msdaora properties=("User ID"=testuser
                   Password=testpass
     "Data Source"="Oraserver"));
  select * from connection to oledb
       (OLEDB::PROVIDER_INFO());
quit;
```

Output 1.1 Provider and Properties Output

```
PROVIDER_NAME    PROVIDER_DESCRIPTION    PROVIDER_PROPERTIES
-------------    --------------------    -------------------
MSDAORA          Microsoft OLE DB        Password;User ID;Data
                 Provider for Oracle     Source;Window Handle;Locale
                                         Identifier;OLE DB Services;
                                         Prompt; Extended Properties;

SampProv         Microsoft OLE DB        Data Source;Window Handle;
                 Sample Provider         Prompt;
```

You could then reference the output when automating a connection to the provider. For the previous result set, you could write the following SAS/ACCESS LIBNAME statement:

```
libname mydblib oledb provider=msdaora
   props=('Data Source'=OraServer 'User ID'=scott 'Password'=tiger);
```

Passing SAS Functions to OLE DB

The engine for OLE DB passes the following operations to OLE DB for processing. See the section about optimizing SQL usage in *SAS/ACCESS for Relational Databases: Reference* for information.

ABS

ARCOS (ACOS)

ARSIN (ASIN)

ATAN

AVG

BYTE

CEIL

COMPRESS

COS

COUNT

DATE

DATEPART

DATETIME

DAY

EXP

FLOOR

HOUR

INDEX

LENGTH

LOG

LOG10

LOWCASE

MAX

MIN

MINUTE

MOD

MONTH

QRT

REPEAT

SECOND

SIGN

SIN

SOUNDEX

SQRT

SUMSTR

SUM

TAN

TIME

TIMEPART

TODAY

TRANWRD

TRIMN

UPCASE

WEEKDAY

YEAR

Passing Joins to OLE DB

In order for a multiple libref join to pass to OLE DB, all of the following components of the LIBNAME statements must match exactly:

- user ID
- password
- datasource
- provider
- qualifier (if specified)
- provider string (if specified)

- UDL_FILE=
  (if specified)
- INIT_STRING=
  (if specified)
- READ_ISOLATION_LEVEL=
  (if specified)
- UPDATE_ISOLATION_LEVEL=
  (if specified)
- all properties specified in the PROPERTIES=() option
- PROMPT=
  must *not* be specified

See the section about performance considerations in *SAS/ACCESS for Relational Databases: Reference* for more information about when and how SAS/ACCESS passes joins to the DBMS.

Temporary Table Support for OLE DB

See the section on the temporary table support in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

Establishing a Temporary Table

When you want to use temporary tables that persist across SAS procedures and DATA steps with OLE DB, you must use the CONNECTION=SHARED LIBNAME option. In doing so, the temporary table is available for processing until the libref is closed.

Terminating a Temporary Table

You can drop a temporary table at any time, or allow it to be implicitly dropped when the connection is terminated. Temporary tables do not persist beyond the scope of a singe connection.

Examples

Using the Internat sample table, the following example creates a temporary table, #LONDON, with Microsoft SQL Server that contains information about flights that flew to London. This table is then joined with a larger SQL Server table that lists all the flights, March, but matched only on flights that flew to London.

```
libname samples oledb Provider=SQLOLEDB Password=dbigrp1 UID=dbitest
                      DSN='lupin\sql2000' connection=shared;

data samples.'#LONDON'n;
   set work.internat;
   where dest='LON';
run;

proc sql;
   select b.flight, b.dates, b.depart, b.orig
```

```
          from samples.'#LONDON'n a, samples.march b
          where a.dest=b.dest;
quit;
```

In the following example a temporary table called New is created with Microsoft SQL Server. The data from this table is then appended to an existing SQL Server table named Inventory.

```
libname samples oledb provider=SQLOLEDB dsn=lupinss
                     uid=dbitest pwd=dbigrp1;

data samples.inventory(DBTYPE=(itemnum='char(5)' item='varchar(30)'
                      quantity='numeric'));
   itemnum='12001';
      item='screwdriver';
      quantity=15;
      output;
   itemnum='12002';
      item='hammer';
      quantity=25:
      output;
   itemnum='12003';
      item='sledge hammer';
      quantity=10;
      output;
   itemnum='12004';
      item='saw';
      quantity=50;
      output;
   itemnum='12005';
      item='shovel';
      quantity=120;
      output;
run;

data samples.'#new'n(DBTYPE=(itemnum='char(5)' item='varchar(30)'
                    quantity='numeric'));
   itemnum='12006';
      item='snow shovel';
      quantity=5;
      output;
   itemnum='12007';
      item='nails';
      quantity=500;
      output;
run;

proc append base=samples.inventory data=samples.'#new'n;
run;

proc print data=samples.inventory;
run;
```

The following example demonstrates the use of a temporary table using the Pass-Through Facility.

```
proc sql;
   connect to oledb as test (provider=SQLOLEDB dsn=lupinss
                             uid=dbitest pwd=dbigrp1);
   execute (create table #FRANCE (flight char(3), dates datetime,
            dest char(3))) by test;

   execute (insert #FRANCE select flight, dates, dest from internat
            where dest like '%FRA%') by test;
   select * from connection to test (select * from #FRANCE);
quit;
```

OLE DB Bulk Loading

The LIBNAME option BULKLOAD= calls the SQLOLEDB interface of IRowsetFastLoad, which enables you to efficiently insert rows of data into a Microsoft SQL Server database table as a unit. BCP= is an alias for this option.

Note: This functionality is available only when accessing Microsoft SQL Server data on Windows platforms using Microsoft SQL Server Version 7.0 or later. △

As SAS/ACCESS sends rows of data to the bulk load facility, the data is written to an input buffer. When you have sent all the rows or when the buffer reaches a certain size (as determined by the DBCOMMIT= option), all of the rows are inserted as a unit into the table and the data is committed to the table. Alternatively, you can set the DBCOMMIT= option to commit rows after a specified number of insertions.

If an error occurs, a message is written to the SAS log, and any rows that were inserted before the error are rolled back.

If you specify BULKLOAD=YES and the PROVIDER= option is set, the SAS/ACCESS interface for OLE DB uses the specified provider. If you specify BULKLOAD=YES and PROVIDER= is not set, the engine assumes the value PROVIDER=SQLOLEDB.

If you specify BULKLOAD=YES, connections that are made through OLE DB Services or UDL files are not allowed.

Locking in the OLE DB Interface

The following LIBNAME and data set options enable you to control how the interface to OLE DB handles locking. See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for additional information about these options.

READ_LOCK_TYPE= ROW | NOLOCK

UPDATE_LOCK_TYPE= ROW | NOLOCK

READ_ISOLATION_LEVEL= S | RR | RC | RU
The default value is set by the data provider. OLE DB supports the S, RR, RC, and RU isolation levels defined in the following table:

Table 1.3 Isolation Levels for OLE DB

Isolation Level	Definition
S (serializable)	Does not allow dirty reads, nonrepeatable reads, or phantom reads.
RR (repeatable read)	Does not allow dirty reads or nonrepeatable reads; does allow phantom reads.
RC (read committed)	Does not allow dirty reads or nonrepeatable reads; does allow phantom reads.
RU (read uncommitted)	Allows dirty reads, nonrepeatable reads, and phantom reads.

The terms in the table are defined as follows:

- *Dirty read* — A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it can see changes that are made by those concurrent transactions even before they commit.

  For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.
- *Nonrepeatable read* — If a transaction exhibits this phenomenon, it is possible that it might read a row once and if it attempts to read that row again later in the course of the same transaction, the row might have been changed or even deleted by another concurrent transaction. Therefore, the read is not (necessarily) repeatable.

  For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.
- *Phantom reads* — When a transaction exhibits this phenomemon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

  For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it sees a row that did not previously exist, a phantom.

UPDATE_ISOLATION_LEVEL= S | RR | RC
The default value is set by the data provider. OLE DB supports the S, RR, and RC isolation levels defined in the preceding table. The RU isolation level is not allowed with this option.

Accessing OLE DB for OLAP Data

Overview

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, or you can create an SQL view of the data. If your MDX statement specifies a data set with more than 5 axes

(COLUMNS, ROWS, PAGES, SECTIONS and CHAPTERS), SAS returns an error. Refer to the Microsoft Data Access Components Software Developer's Kit for details about MDX syntax.

Note: This implementation provides read-only access to OLE DB for OLAP data. You cannot update or insert data with this facility. △

Using the Pass-Through Facility with OLAP Data

The main difference between normal OLE DB access using the Pass-Through Facility and the implementation for OLE DB for OLAP is the use of additional identifiers to pass MDX statements to the OLE DB for OLAP data. These identifiers are the following:

MDX::
identifies MDX statements that return a flattened data set from the multidimensional data.

MDX_DESCRIBE::
identifies MDX statements that return detailed column information.

An MDX_DESCRIBE:: identifier is used to obtain detailed information about each returned column. During the process of flattening multidimensional data, OLE DB for OLAP builds column names from each level of the given dimension. For example, for OLE DB for OLAP multidimensional data that contains CONTINENT, COUNTRY, REGION, and CITY dimensions, you could build a column with the following name:

```
[NORTH AMERICA].[USA].[SOUTHEAST].[ATLANTA]
```

This name cannot be used as a SAS variable name because it has more than 32 characters. For this reason, the SAS/ACCESS engine for OLE DB creates a column name based on a shortened description, in this case, ATLANTA. However, since there could be an ATLANTA in some other combination of dimensions, you might need to know the complete OLE DB for OLAP column name. Using the MDX_DESCRIBE:: identifier returns a SAS data set that contains the SAS name for the returned column and its corresponding OLE DB for OLAP column name:

```
SASNAME          MDX_UNIQUE_NAME

ATLANTA          [NORTH AMERICA].[USA].[SOUTHEAST].[ATLANTA]
CHARLOTTE        [NORTH AMERICA].[USA].[SOUTHEAST].[CHARLOTTE]
   .                        .
   .                        .
   .                        .
```

If two or more SASNAME values are identical, a number is appended to the end of the second and later instances of the name; for example, ATLANTA, ATLANTA0, ATLANTA1, and so on. Also, depending on the value of the VALIDVARNAME= system option, illegal characters are converted to underscores in the SASNAME value.

Syntax

This facility uses the following general syntax. For more information about Pass-Through Facility syntax, see in *SAS/ACCESS for Relational Databases: Reference*.

PROC SQL <*options*>;

CONNECT TO OLEDB (<*options*>);
<*non-SELECT SQL statement(s)*>
SELECT *column-identifier(s)* FROM CONNECTION TO OLEDB
(MDX:: | MDX_DESCRIBE:: <*MDX statement*>)
<*other SQL statement(s)*>
;

Examples

The following code uses the Pass-Through Facility to pass an MDX statement to a Microsoft SQL Server Decision Support Services (DSS) Cube. The provider used is the Microsoft OLE DB for OLAP provider named MSOLAP.

```
proc sql noerrorstop;
   connect to oledb (provider=msolap prompt=yes);
   select * from connection to oledb
      ( MDX::select {[Measures].[Units Shipped],
             [Measures].[Units Ordered]} on columns,
             NON EMPTY [Store].[Store Name].members on rows
              from Warehouse );
```

See the Microsoft Data Access Components Software Developer's Kit for details about MDX syntax.

The CONNECT statement requests prompting for connection information, which facilitates the connection process (especially with provider properties). The MDX:: prefix identifies the statement within the parentheses that follows the MDX statement syntax, and is not an OLAP-specific SQL statement. Partial output from this query might look like this:

```
Store               Units Shipped         Units Ordered

Store6              10,647                11,699
Store7              24,850                26,223
   .                   .                     .
   .                   .                     .
   .                   .                     .
```

You can use the same MDX statement with the MDX_DESCRIBE:: identifier to see the full description of each column:

```
proc sql noerrorstop;
connect to oledb (provider=msolap prompt=yes);
select * from connection to oledb
   ( MDX_DESCRIBE::select {[Measures].[Units Shipped],
                  [Measures].[Units Ordered]} on columns,
                  NON EMPTY [Store].[Store Name].members on rows
                  from Warehouse );
```

The next example creates a view of the OLAP data, which is then accessed using the PRINT procedure:

```
proc sql noerrorstop;
  connect to oledb(provider=msolap
                  props=('data source'=sqlserverdb
                     'user id'=myuserid password=mypassword));
  create view work.myview as
     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]}),
               [Measures].[Unit Sales],DESC) on rows
             from Sales )
;

proc print data=work.myview;
run;
```

In this example, full connection information is provided in the CONNECT statement, so the user is not prompted. The SQL view can be used in other PROC SQL statements, the DATA step, or in other procedures, but you cannot modify (that is, insert, update, or delete a row in) the view's underlying multidimensional data.

Naming Conventions for OLE DB

Because OLE DB is an application programming interface (API), data source names for files, tables, and columns are determined at run time. In Version 7 and later, most SAS names can be up to 32 characters long. The SAS/ACCESS interface for OLE DB also supports file, table, and column names up to 32 characters long. If the column names are longer than 32 characters, they are truncated to 32 characters. If truncating a name results in identical names, then SAS generates unique names by replacing the last character with a number. For more information, see the section about SAS names and support for DBMS names in *SAS/ACCESS for Relational Databases: Reference*.

The PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options determine how the interface to OLE DB handles case sensitivity, spaces, and special characters. The default value for both options is NO for most data sources. The default value is YES for Microsoft Access, Microsoft Excel, and Microsoft SQL Server.

Data Types for OLE DB

Each data source column in a table has a name and a data type. The data type tells the data source how much physical storage to set aside for the column and the form in which the data is stored.

The following table shows all of the data types and default SAS formats that are supported by the SAS/ACCESS interface to OLE DB. This table does not explicitly define the data types as they exist for each data source. It lists the types that each data source's data type might map to. For example, an INTEGER data type under DB2 might map to an OLE DB data type of DBTYPE_I4. All data types are supported.

Table 1.4 OLE DB Data Types and Default SAS Formats

OLE DB Data Type	Default SAS Format
DBTYPE_R8	none
DBTYPE_R4	none
DBTYPE_I8	none
DBTYPE_UI8	none
DBTYPE_I4	11.

OLE DB Data Type	Default SAS Format
DBTYPE_UI4	11.
DBTYPE_I2	6.
DBTYPE_UI2	6.
DBTYPE_I1	4.
DBTYPE_UI1	4.
DBTYPE_BOOL	1.
DBTYPE_NUMERIC	*m* or *m.n* or none, if *m* and *n* are not specified
DBTYPE_DECIMAL	*m* or *m.n* or none, if *m* and *n* are not specified
DBTYPE_CY	DOLLAR*m*.2
DBTYPE_BYTES	$*n*.
DBTYPE_STR	$*n*.
DBTYPE_BSTR	$*n*.
DBTYPE_WSTR	$*n*.
DBTYPE_DBDATE	DATE9.
DBTYPE_DBTIME	TIME8.
DBTYPE_DBTIMESTAMP and DBTYPE_DATE	DATETIME*m.n*, where *m* depends on precision and *n* depends on scale

The following table shows the default data types that the SAS/ACCESS interface to OLE DB uses when creating DBMS tables.

Table 1.5 Default OLE DB Output Data Types

SAS Variable Format	Default OLE DB Data Type
m.n	DBTYPE_R8 or DBTYPE_NUMERIC using *m.n* if the DBMS allows it
$*n*.	DBTYPE_STR using *n*
date formats	DBTYPE_DBDATE
time formats	DBTYPE_DBTIME
datetime formats	DBTYPE_DBTIMESTAMP

The SAS/ACCESS interface to OLE DB allows nondefault data types to be specified with the DBTYPE= data set option.

OLE DB Null Values

Many relational database management systems have a special value called NULL. A DBMS NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a DBMS NULL value, it interprets it as a SAS missing value.

In most relational databases, columns can be defined as NOT NULL so that they require data (they cannot contain NULL values). When a column is defined as NOT NULL, the DBMS will not add a row to the table unless the row has a value for that

column. When creating a DBMS table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

OLE DB mirrors the behavior of the underlying DBMS with regard to NULL values. Refer to the documentation for your DBMS for information about how it handles NULL values.

For more information about how SAS handles NULL values, see "Potential Result Set Differences When Processing Null Data" in *SAS/ACCESS for Relational Databases: Reference*.

Note: To control how SAS missing character values are handled by the DBMS, use the NULLCHAR= and NULLCHARVAL= data set options. △