CHAPTER

1

OLE DB Chapter, First Edition

Introduction

SAS/ACCESS software enables you to access data in a database management system (DBMS) or other data source in order to use that data in your SAS programs. Microsoft OLE DB is an *API* (application programming interface) that provides access to data, which can be in many forms, including a database table, an email file, a text file, or other kind of file. This SAS/ACCESS interface accesses data from these sources through the OLE DB data providers. You specify the data provider, data source, and other connection information in a SAS/ACCESS LIBNAME statement and the SQL Procedure Pass-Through facility. The SQL procedure is a base SAS procedure that works with SAS/ACCESS software to send and receive data directly between a data source and SAS software. You can store Pass-Through code in a PROC SQL view for later use.

For more information about the OLE DB API, see the Microsoft OLE DB reference documentation. This chapter accompanies and should be used with *SAS/ACCESS Software for Relational Databases: Reference* (order #57204).*

SAS/ACCESS LIBNAME Statement

The SAS/ACCESS LIBNAME statement and options that can be used in most relational databases are fully described in Chapter 3, "SAS/ACCESS LIBNAME Statement". This section describes the connection options for OLE DB and any OLE DB-specific LIBNAME options.

LIBNAME Statement: OLE DB Specifics

Associates a SAS libref with a data source

Valid: in a DATA or PROC step

Syntax

LIBNAME *libref SAS/ACCESS-engine-name*
SAS/ACCESS-engine-connection-options
< *SAS/ACCESS-LIBNAME-options*>;

Arguments

libref
is any SAS name that serves as an alias to associate the SAS System with a data source.

SAS/ACCESS-engine-name
is a SAS/ACCESS engine name for your DBMS or data source, in this case, **OLEDB**. SAS/ACCESS engines are implemented differently in different operating environments. The engine name is required.

SAS/ACCESS-engine-connection-options
are options that you specify in order to connect to a particular data source; these options are different for each data source. If the SAS/ACCESS engine connection options contain characters that are not allowed in SAS names, enclose the values of the options in quotation marks. If you specify the appropriate system options or environment variables for your data source, you can often omit the SAS/ACCESS engine connection options.

SAS/ACCESS-LIBNAME-options
are options that apply to a data source, such as a database table. For example, the STRINGDATES= option specifies whether to read date and time values as character strings or as numeric date values. Support for many of the LIBNAME options is specific to OLE DB data providers.

Some SAS/ACCESS LIBNAME options have the same names as SAS/ACCESS-engine data-set options. When you specify an option in the LIBNAME statement, it applies to the particular data source (which is accessed by the libref). A SAS/ACCESS data set option applies only to the data set on which it is specified. If a like-named option is specified in both the SAS/ACCESS LIBNAME statement and after a data set name (which represents a data source table or view), the SAS System uses the value that is specified after the data set name. For more information, see Chapter 3, "SAS/ACCESS LIBNAME Statement".

Details The LIBNAME statement associates a libref with a SAS/ACCESS engine in order to access a data source. The SAS/ACCESS engine enables you to connect to a particular data source and, therefore, to specify a DBMS table or other file name in a two-level SAS name.

For example, in **mydblib.q2_employees**, **mydblib** is a SAS libref that points to a particular data source, and **q2_employees** is a DBMS table name. When you specify **mydblib.q2_employees** in a DATA step or procedure, you dynamically access that

DBMS table. Version 8 the SAS System supports reading, updating, creating, and deleting DBMS tables. (However, it does not support altering a database by adding or deleting a column. For these tasks, you must use the DBMS-specific SQL statements.)

See for more information about arguments and options that you can use in the LIBNAME statement.

Connecting to OLE DB There are many SAS/ACCESS connection options available when you use the SAS/ACCESS interface to OLE DB, and these options have several interactions. Basically, there are two separate ways to connect to a data source: using OLE DB Services or connecting directly to the provider. Each has its advantages and its limitations, as described here.

Connecting with OLE DB Services Often the fastest and easiest way to connect to a data provider is by using OLE DB Services. Examples of data providers include Microsoft Access, Microsoft SQL Server, or ORACLE. For each of the listed providers, the data source would be a relational database that contains database objects, such as tables and views, whose data you could then access in your SAS programs. OLE DB accepts other sources of data as well, including email files and text files.

OLE DB Services provides some performance optimizations and scaling features, including resource pooling. It also displays dialog windows to prompt you for connection information, where the dialogs are consistent regardless of which provider you use.

As described in “Connecting Directly with the Data Provider” on page 5, you can specify the SAS/ACCESS connection options, PROVIDER= and PROPERTIES=, to connect to a data provider and its data sources. In PROPERTIES=, you supply the connection information; for example, if the data source were a relational database, you might supply the user ID, password,and schema. The default action for SAS/ACCESS software is to use OLE DB Services, so you can omit the OLEDB_SERVICES= option and still get the benefits of the Services.

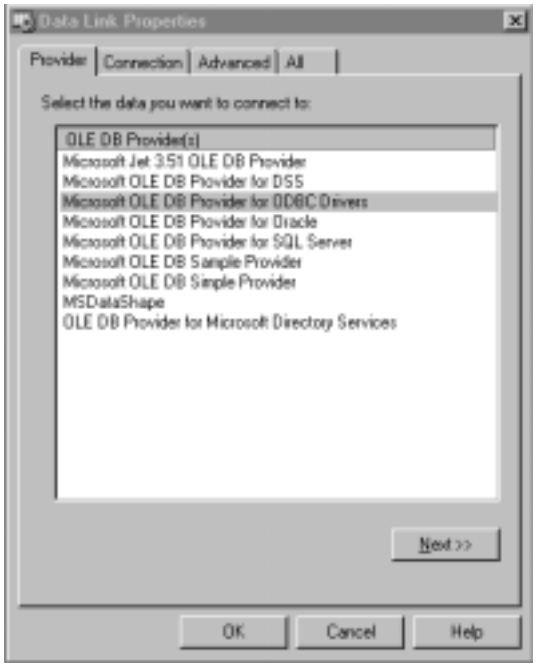
Using OLE DB Services enables you to be prompted for the provider name and properties during an interactive SAS session. To be prompted, submit your libref and a SAS/ACCESS LIBNAME statement for OLE DB:

```
libname mydblib oledb;
```

In this case, the SAS/ACCESS engine for OLE DB assumes that you want to be prompted (and therefore sets PROMPT=YES). It directs OLE DB Services to display the prompting dialog windows so that you can supply the provider name and connection information for the data source. If you choose, you can specify the PROPERTIES= option in the LIBNAME statement, supply some of the connection information, and still be prompted to supply the rest of the information in the dialog windows.

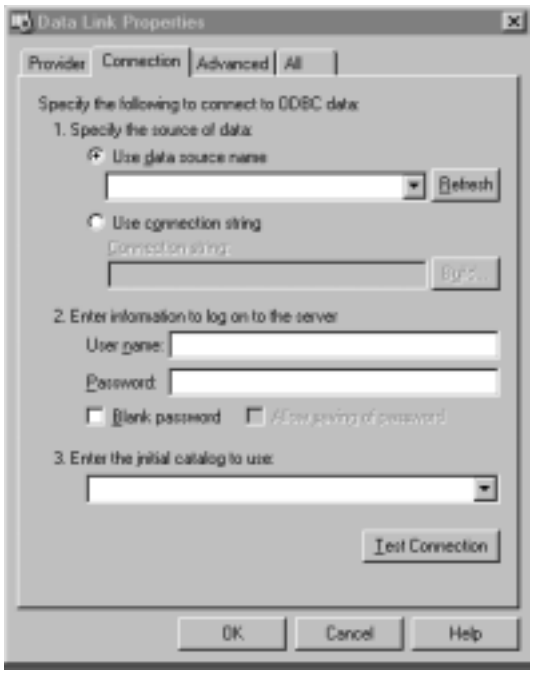
Display 1.1 on page 4 shows you the initial prompting dialog window and some of the providers from which you can choose:

Display 1.1 OLE DB Services Dialog Window, First Tab



Display 1.2 on page 4 shows where you enter the data source name and other server information:

Display 1.2 OLE DB Services Dialog Window, Second Tab



After you have been prompted for the connection information and made a successful connection, the SAS/ACCESS engine for OLE DB and OLE DB Services enable you to retrieve the connection information and re-use it in batch jobs and automated

connections. For details, see the SAS/ACCESS connection option, INIT_STRING= on page 6. See also the option, OLEDB_SERVICES= on page 7.

Connecting Directly with the Data Provider If you know the data provider's name and properties, you can connect directly to the provider and its data source(s). Examples of data providers include Microsoft Access and ORACLE, where the sources of data that you might access could be tables in a relational database, email files, or text files.

OLE DB does not have an administrator, like the ODBC Data Source Administrator, to help you create and list data sources. Therefore, you have to determine the data providers and the properties that they accept. To make the connection to the provider's data source, SAS requires connection information, such as the user ID, password, schema, and server name.

If you connect directly to the provider, your minimum set of SAS/ACCESS connection options are PROVIDER=, and OLEDB_SERVICES=NO, assuming that you have default values set for the provider's properties. (The default action for OLEDB_SERVICES= is YES, so you have to indicate that you do not want to use the OLE DB Services. For more information about OLE DB Services, see "Connecting with OLE DB Services" on page 3.)

After you connect to your provider once (for example, by using OLE DB Services or through a prompted session), you can use a special OLE DB query called PROVIDER_INFO()"Special OLE DB Queries" on page 33 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" on page 36.

If you are connecting to MS SQL Server and specifying the SAS/ACCESS option BCP=YES, then you must connect directly to the provider.

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. The provider displays its own dialog window for prompting; these dialogs vary from provider to provider.

In addition to the SAS/ACCESS connection options already listed, you can specify the SAS/ACCESS options, COMPLETE= and REQUIRED=. Use these options to try to connect to the data source using the information that you have already supplied with a PROPERTIES= options. If more information is needed to make the connection, the dialog windows prompt you for it.

If you run SAS in a batch environment, you must supply the provider name and all of the connection information in the PROPERTIES= option. In this case, you specify only the following SAS/ACCESS connection options: PROVIDER=, PROPERTIES=, and OLEDB_SERVICES=NO.

The following table summarizes how you can use the SAS/ACCESS-engine connection options:

Table 1.1 Two Ways to Connect OLE DB

	Minimum options required	All options allowed
Connect through OLE DB Services	none	provider=, oledb_services=yes properties=, provider_string=, init_string=, prompt=
Connect directly to OLE DB	provider= * , oledb_services=no	provider=, oledb_services=no, properties=, provider_string=, prompt=, complete=, required=

* You can omit the PROPERTIES= option if you have default properties set for your provider.

SAS/ACCESS-Engine Connection Options The SAS/ACCESS-engine connection options are as follows:

COMPLETE=YES | NO
specifies whether the SAS/ACCESS engine for OLE DB tries to connect to the data source with the properties that you specified in PROPERTIES=. If COMPLETE=YES, the engine tries to connect. If enough information is specified for a successful connection, then the connection is made without any prompting for more information.

Default value: NO.

If you specify COMPLETE= NO or do not have enough information to connect, you are prompted for the connection options with a dialog window.

COMPLETE= is used only when you connect directly to OLE DB using the PROVIDER= and OLEDB_SERVICES=NO in a SAS/ACCESS LIBNAME statement.

See also these options: PROPERTIES= on page 8, PROMPT=YES on page 7, PROVIDER= on page 8, OLEDB_SERVICES= on page 7.

INIT_STRING="<*initilization-string*>"
using OLE DB Services, INIT_STRING= specifies an initialization string when connecting to a data source. After you are prompted to supply information to connect to your data source, the SAS/ACCESS engine for OLE DB returns the complete initialization string to the macro variable, SYSDBMSG. You can then re-use the initialization string to connect to the same provider and data source.

Default value: option is not set. However, if you specify the option with a null argument, (**`INIT_STRING=""`**), OLE DB connects to ODBC with a default set of properties. See the Microsoft OLE DB documentation for more information about these default values.

In this example, you submit the basic connection information so that you will be prompted for the rest of the information. Using OLE DB Services is the default value, so you can omit the OLEDB_SERVICES= option.

```
libname mydblib oledb;
```

Through dialog windows, OLE DB Services prompts you for the provider and properties' values. The advantage of being prompted is that you do not need to know any special syntax to set the properties'. Prompting also enables you to set more options than you might when connecting directly to the provider (and not using OLE DB Services).

After connecting to the data source, the SAS/ACCESS engine for OLE DB returns the initialization string to the SYSDBMSG macro variable. To write this

string to the SAS log *immediately* after connecting to the data source, submit the following:

```
%put &SYSDBMSG;

OLEDB:  Provider=SQLOLEDB;Password=dbmgr1;
Persist Security Info=True;User ID=rachel;
Initial Catalog=users;Data Source=DBPC6;
```

The SYSDBMSG information mirrors all of the options that you chose during your prompted connection. Notice that the initialization string is prefixed with **OLEDB:**. When you store the string for later use, delete this prefix and any initial spaces before the first listed option.

By storing the initialization string, you can re-use it in the INIT_STRING= option to make automated connections or to specify this option in batch jobs:

```
init_string="Provider=SQLOLEDB;Password=dbmgr1;Persist
Security Info=True;User ID=rachel;Initial Catalog=users;
Data Source=DBPC6";
```

Using INIT_STRING= enables you to bypass the prompting window but still gives you the advantages of the OLE DB Services, such as performance optimizations.

Specifying INIT_STRING= overrides any values that were set with the SAS/ACCESS connection options PROVIDER= on page 8 and PROPERTIES= on page 8.

Alias: INIT=.

OLEDB_SERVICES=YES | NO
determines whether the SAS/ACCESS engine for OLE DB uses OLE DB Services. OLEDB_SERVICES=YES causes the engine to use OLE DB Services, and OLEDB_SERVICES=NO causes the engine to use the provider to connect to the data source.

Default value: YES.

Generally, OLE DB Services is easier to use and more consistent. When OLEDB_SERVICES=YES and a successful prompted connection is made, the complete connection string is returned in the SYSDBMSG macro variable. For more information, see the option INIT_STRING= on page 6.

OLEDB_SERVICES= interacts with other connection options. If you have set PROMPT=YES, OLEDB_SERVICES=YES enables you to set more options than you would be able to set by being prompted by the provider's dialog window. If OLEDB_SERVICES=NO, you must specify PROVIDER= first in order for the provider's prompt dialogs to be used. If PROVIDER= is omitted, the SAS/ACCESS engine uses OLE DB Services, regardless of how the OLEDB_SERVICES= option is set.

If the BCP=YES option is set for Microsoft SQL Server data, then OLEDB_SERVICES=NO. OLEDB_SERVICES= also interacts with the PROMPT=, REQUIRED= , and COMPLETE= options. See these options for more information: PROVIDER= on page 8, BCP=YES on page 10, PROMPT= on page 7, REQUIRED= on page 9, COMPLETE= on page 6.

PROMPT =YES | NO
enables you to be prompted for connection information to supply to the data source. The kind of prompting that you receive depends on how you set the PROVIDER= and OLEDB_SERVICES= options.

Default value: NO, if the provider is specified; otherwise, YES.

If a provider name is specified *and* OLEDB_SERVICES= NO, the OLE DB provider displays a dialog window that contains the connection information and

property attributes. If the provider name is omitted *or* OLEDB_SERVICES=YES, the OLE DB Services displays a dialog window that enables you to select a provider and to specify connection information and property attributes. The dialog window for OLE DB Services is generally preferred over the provider's dialog window because the OLE DB Services window enables you to set options more easily.

If PROMPT=YES, properties that were set with PROPERTIES= will be displayed in the dialog window. This applies both to the provider dialog window and to the OLE DB Services dialog window. You can edit any field before you connect to the data source.

If the provider name is omitted, the SAS/ACCESS engine for OLE DB tries to prompt you for the connection information by using the OLE DB Services dialog window. This applies even if PROMPT=NO and OLEDB_SERVICES= NO. If the provider name is omitted in batch mode, the connection fails.

If you are unsure what to specify for various provider properties, use the PROMPT= option to guide you through the connection process.

See the following options for more information: PROVIDER= on page 8,OLEDB_SERVICES= on page 7, PROPERTIES= on page 8.

PROPERTIES=*(<">property-name-1<">=<">property-value-1<"> . . . <">property-name-n<">=<">property-value-n<">)*

specifies provider properties that enable you to connect to a data source and to define the attributes of that connection. Each property name is assigned a value using an equal sign (=). If the property name or value contains embedded spaces or special characters, enclose the name in double quotes. Separate multiple pairs with a space. PROPERTIES= is optional in OLE DB.

Default value: none.

In this example, you specify a user ID and password to connect to a Microsoft SQL Server data source, you would enter:

```
libname mydblib oledb provider=sqloledb
                properties=("User ID"=shala
                Password="mypw@hr");
```

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

Aliases: PROPS= and PROP=.

PROVIDER=<'> *your-provider-name*<'>

specifies the OLE DB provider to use in order to connect to the data source. The PROVIDER= option is required during batch processing.

Default value: none.

There is no restriction on the length of the name. Put names with non-standard SAS characters (such as spaces, colons, @ signs) in single or double quotations marks.

If you omit this option, you are prompted for the provider name. It is recommended that, if possible, you use the dialog prompts to connect to your data source. The prompts enable you to use an interactive interface to enter the name of the provider, properties, and connection options.

Alias: PROV=.

PROVIDER_STRING=<">*provider-name*<">

passes additional provider-specific connection information to the provider. The provider-string is enclosed in quotation marks if it contains blank spaces or special

characters. The PROVIDER_STRING= option works whether you connect directly to the provider or are using OLE DB Services.

Default value: none

Microsoft uses a provider-string for its Jet provider in order to determine the type of data source to which it connects. MS Jet currently accepts the following providers; this list is not all-inclusive and is subject to change by Microsoft:

dBase III, IV, 5.0

Excel 3.0, 4.0, 5.0, 8.0

Exchange 4.0

HTML Export, HTML Import

Jet 2.*x*, Jet 3.*x*

Lotus WJ2, WJ3

Lotus WK1, WK2, WK3, WK4

Outlook 9.0

Paradox 3.*x*, Paradox 4.*x*, Paradox 5.*x*, Paradox 7.*x*

Text

Providers other than MS Jet also accept other provider-strings.

In the following SAS/ACCESS LIBNAME statement example, you use the Microsoft Jet 4.0 provider to access the spreadsheet **Y2Kbudgetworksheet.xls**. Notice that you must specify the provider-string **"Excel 8.0"** so that MS Jet recognizes the file as an Excel 8.0 worksheet.

```
libname y2kbudget oledb provider="Microsoft.Jet.OLEDB.4.0"
  properties=('data source'='d:\excel80\Y2Kbudgetworksheet.xls')
  provider_string="Excel 8.0";
```

REQUIRED=YES | NO

indicates whether you specify connection options for your provider.

Default value: NO.

If you specify REQUIRED= YES, the SAS/ACCESS engine for OLE DB tries to connect to the data source by using the properties that were specified in PROPERTIES= . If you specify enough information in order to make a connection, then the connection is made without prompting. Otherwise, a dialog window is displayed to prompt you for the connection options. Options in the dialog window that are not related to the connection are disabled.

REQUIRED= is used only when you connect to the provider directly using the options PROVIDER= and OLEDB_SERVICES=NO in a SAS/ACCESS LIBNAME statement.

For more information, see PROPERTIES= on page 8, OLEDB_SERVICES= on page 7, and PROVIDER= on page 8.

SAS/ACCESS LIBNAME Options When you specify any of the following options on the LIBNAME statement, the option is applied to the data source that the libref represents. When the data source is a relational database, the LIBNAME option applies to all of the objects in the database (such as its tables, views, and indexes) that the libref represents.

The SAS/ACCESS interface to OLE DB supports all of the SAS/ACCESS LIBNAME options listed in Chapter 3, "SAS/ACCESS LIBNAME Statement" , except for

DBPROMPT=. In addition to the supported options, the following LIBNAME options are used only in the interface to OLE DB or have OLE DB–specific aspects to them:

AUTOCOMMIT=YES | NO
indicates whether or not updates are committed immediately after they are submitted. This option applies if your data source is a relational database. This is a LIBNAME-only option.

Default value: NO.

If AUTOCOMMIT=YES, updates are committed (that is, saved) to table as soon as they are submitted, and no rollback is possible. If AUTOCOMMIT=NO, the SAS/ACCESS engine automatically performs the commit when it reaches the end of the file.

For the SAS/ACCESS LIBNAME engine, the default value is NO, if the data source provider supports transactions and it is a connection for updating data. Otherwise, the default value is AUTOCOMMIT= YES (that is, YES for the SQL Procedure Pass-Through Facility and read-only connections).

BCP=YES | NO
determines whether SAS uses the Microsoft Bulk Copy facility to insert data into a DBMS table. Specify BCP=YES to direct SAS to use the OLE DB BCP interface when inserting data into a Microsoft SQL Server database table. (MS SQL Server 7.0 and later provides this support.) The BCP= option is valid only in a SAS/ACCESS LIBNAME statement when connecting to MS SQL Server.

Default value: NO.

BCP is Microsoft's Bulk Copy facility, and it enables you to efficiently insert rows of data into a DBMS table as a unit. As SAS/ACCESS sends each row of data to BCP, the data is written to an input buffer. When you have inserted all the rows or the buffer reaches a certain size (as determined by the DBCOMMIT= data

set 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=*n* option to commit rows after every *n* insertions. See Chapter 4, "SAS/ACCESS Data Set Options".

If an error occurs, a message is written to the SAS log, and any rows that have been inserted in the table before the error will be rolled back.

If you specify BCP=YES and the PROVIDER= option is set, the SAS/ACCESS engine for OLE DB uses the specified provider. If you specify BCP=YES and PROVIDER= is not set, the engine assumes the value PROVIDER=SQLOLEDB. If you specify BCP=YES, connections that are made through OLE DB Services are not allowed; that is, specifying BCP=YES means that OLEDB_SERVICES=NO.

CELLPROP=<'>*value*<'> | *'formatted-value'*
modifies the metadata and content of a result data set that is defined through an MDX command. When an MDX command is issued, the resulting data set might have columns that contain one or more types of data values: the actual value of the cell or the formatted value of the cell. (A *cell* or data value refers to the intersection of a column and a row.)

Default value: <'>*value*<'>.

For example, if you were to issue an MDX command and the resulting data set contained a column named **SALARY**, the column could contain data values of two types. It could contain numeric values, such as **50000**, or it could contain formatted values, such as **$50,000**. Setting the CELLPROP= option determines how the values are defined and the value of the column.

If CELLPROP=<'>*value*<'>, the SAS/ACCESS engine for OLE DB tries to return the actual data value; if the value is numeric, then the column is defined as NUMERIC.

If CELLPROP= *'formatted-value'*, the SAS/ACCESS engine for OLE DB defines the column as CHARACTER and it returns the formatted data values.

It is possible for a column in a result set to contain both NUMERIC and CHARACTER data values. For example, a data set might return the data values of **50000**, **60000**, and **UNKNOWN**. SAS data sets cannot contain both types of data. Therefore, even if you specify CELLPROP=<'>*value*<'>, the SAS/ACCESS engine defines the problematic column as CHARACTER and returns formatted values for that column.

For more information about MDX commands, see “Accessing OLE DB for OLAP Data” on page 37.

COMMAND_TIMEOUT=*number-of-seconds*
specifies the number of seconds to wait before a data source command times out. This is a LIBNAME and data set option.

Default value: 0 (no timeout).

Alias: TIMEOUT=.

CURSOR_TYPE=DYNAMIC | KEYSET_DRIVEN | STATIC
specifies the cursor type for read only and updatable cursors. This is both a LIBNAME and data set option.

Default value: None.

When your data source is a relational database, not all DBMS drivers support all cursor types. An error is returned if the specified cursor type is not supported.

If CURSOR_TYPE=DYNAMIC, then the cursor reflects all of the changes that are made to the rows in a result set as you scroll around the cursor. The data values and the membership of rows in the cursor can change dynamically on each fetch. The OLE DB properties that are applied to an open row set are

DBPROP_OTHERINSERT=TRUE and
DBPROP_OTHERUPDATEDELETE=TRUE.

If CURSOR_TYPE=KEYSET_DRIVEN, then the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows will be reflected as you scroll around the cursor. The OLE DB properties that are applied to an open row set are
DBPROP_OTHERINSERT=FALSE and
DBPROP_OTHERUPDATEDELET=TRUE.

If CURSOR_TYPE=STATIC, then the cursor builds the complete result set when the cursor is opened. No changes made to the rows in the result set after the cursor is opened will be reflected in the cursor. Static cursors are read-only. The OLE DB properties that are applied to an open row set are
DBPROP_OTHERINSERT=FALSE and
DBPROP_OTHERUPDATEDELETE=FALSE.

By default, CURSOR_TYPE= is not set and the provider will use a default. See your provider documentation for more information. See OLE DB programmer reference documentation for details about these properties.

Alias: CURSOR.

DELETE_MULT_ROW=YES | NO
indicates whether to allow SAS to delete multiple rows from a data source, such as a DBMS table. This is a LIBNAME-only option.

Some providers do not handle the following DBMS SQL statement well, and therefore delete more than the current row with this statement: **DELETE ... WHERE CURRENT OF CURSOR**. DELETE_MULT_ROW= enables SAS/ACCESS to continue if multiple rows were deleted.

Default value: NO

PRESERVE_COL_NAMES=YES | NO
preserves blank spaces, special characters, and mixed case in the column names of a data source, such as a relational database table. This is a LIBNAME-only option.

Default value: NO for most data sources. For the following data sources, the default value is YES: MS Access, MS Excel, and MS SQL Server.

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement".

PRESERVE_TAB_NAMES=YES | NO
preserves blank spaces, special characters, and mixed case in a data source, such as a relational database table. This is a LIBNAME-only option.

Default value: NO for most data sources. For the following data sources, the default value is YES: MS Access, MS Excel, and MS SQL Server.

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement".

QUALIFIER=<*qualifier-name*>
enables you to read a data source using the specified qualifier. Or, when the data source is a relational database, you can read the database objects, such as tables and views, using the specified qualifier. QUALIFIER= is both a LIBNAME and data set option.

QUALIFIER= is optional. If it is omitted, you use the default DBMS qualifier name, if any. QUALIFIER= can be used for any DBMS that allows three-part identifier names, such as *qualifier.schema.object*.

The following LIBNAME statement connects to an MS SQL Server table. The QUALIFIER= option causes any reference in SAS to **mydblib.employee** to be interpreted by OLE DB as **pcdivision.raoul.employee**.

```
libname mydblib oledb provider=SQLOLEDB
    properties=("user id"=dbajorge
    "data source"=SQLSERVR)
    schema=raoul qualifier=pcdivision;
proc print data=mydblib.employee;
run;
```

QUOTE_CHAR=*character*
specifies the quotation character to use when delimiting identifiers, such as double-quote character ("). This is a LIBNAME-only option.

The provider usually specifies the delimiting character. However, when there is a difference between what the provider allows for this character and what the data source allows, the QUOTE_CHAR= option enables you to override the character returned by the provider.

Default value: the option is not set, and it uses the quotation character returned by the provider.

READ_ISOLATION_LEVEL=S | RR | RC | RU
defines the degree of isolation of the current application process from other concurrently running application processes. This is both a LIBNAME and data set option.

Default value: Set by the data provider.

The arguments for READ_ISOLATION_LEVEL= indicate the following: S = Serializable, RR = Repeatable Read, RC = Read Committed, and RU = Read Uncommitted.

OLE DB supports five levels of isolation, which are described in detail in "READ_ISOLATION_LEVEL=" on page 21. The degree of isolation identifies

- the degree to which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree to which update activity of other concurrently executing application processes can affect the current application.

The READ_ISOLATION_LEVEL= option applies only when reading a data source, such as a DBMS table or view. By default, this option is not set. The provider sets the default value. If READ_LOCK_TYPE= is not set to ROW, then READ_ISOLATION_LEVEL= is ignored. See also READ_LOCK_TYPE= on page 13.

Alias: RIL=.

READ_LOCK_TYPE=ROW
specifies how a table is locked during a READ operation.

The value ROW specifies that a row or set of rows will be locked. SAS/ACCESS software uses the READ_ISOLATION_LEVEL= option to determine which rows will be locked. Currently, the only valid locking that is allowed is through the READ_ISOLATION_LEVEL= option; therefore, READ_LOCK_TYPE= only allows a value of ROW. See also READ_ISOLATION_LEVEL= on page 13.

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

ROWSET_SIZE=*number-of-rows*
specifies the number of rows to use when reading data from a data source, such as a DBMS table. This is both a LIBNAME and data set option.

Default value: 1, so that only one row is retrieved at a time.

The higher the value for ROWSET_SIZE=, the more rows that the SAS/ACCESS engine for OLE DB retrieves in one fetch operation. This option reduces the amount of I/O that is used and can help improve performance. However,

because SAS software stores the rows in memory, higher values for ROWSET_SIZE= use more memory. In addition, if too many rows are selected at once, then the rows that are returned to the SAS application might be out of date.

Alias: ROWSET=.

SCHEMA=*schema-name*

enables you to read a data source using the specified schema. Or, when the data source is a relational database, you can read the database objects, such as tables and views, using the specified schema.

SCHEMA= is both a LIBNAME and data set option. SHEMA= is optional. If it is omitted, you connect to the default schema.

In the following LIBNAME statement, the SCHEMA= option causes any reference in SAS to **mydblib.employee** to be interpreted by the OLE DB provider (ORACLE) as **raoul.employee**.

```
libname mydblib oledb provider=msdaora
        properties=("user id"=dbajorge password=dbajorge99
        "data source"="oracle_loc")
        schema=raoul qualifier=pcdivision;
proc print data=mydblib.schedule;
run;
```

Alias: OWNER=.

STRINGDATES=YES | NO

specifies whether to read datetime values from the data source, such as a DBMS table, as character strings, or to read them as numeric date values. This is a LIBNAME-only option.

Default value: NO.

If STRINGDATES=YES, SAS/ACCESS reads datetime values as character strings. If STRINGDATES=NO, SAS/ACCESS reads datetimes values as numeric date values.

Alias: STRDATES.

UPDATE_ISOLATION_LEVEL=S | RC | RR

Defines the degree of isolation of the current application process from other concurrently running application processes.

The arguments for UPDATE_ISOLATION_LEVEL= indicate the following: S = Serializable, RC = Read Committed, and RR = Repeatable Read.

OLE DB supports four isolation levels, which are described in detail in "UPDATE_ISOLATION_LEVEL=" on page 25. The degree of isolation identifies

- the degree to which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree to which update activity of other concurrently executing application processes can affect the current application.

By default, UPDATE_ISOLATION_LEVEL= in not set. The provider sets the default value. If UPDATE_LOCK_TYPE= is not set to ROW, then UPDATE_ISOLATION_LEVEL= is ignored. See also UPDATE_LOCK_TYPE= on page 14.

Alias: UIL=.

UPDATE_LOCK_TYPE=ROW

specifies how a data source, such as a DBMS table, is locked during an UPDATE operation. This is both a LIBNAME and data set option.

Default value: ROW.

The UPDATE_ISOLATION_LEVEL= option applies only when updating a DBMS table or view. The value ROW specifies that a row or set of rows will be

locked. To determine which rows will be locked, SAS/ACCESS software uses the UPDATE_ISOLATION_LEVEL= option. See also UPDATE_ISOLATION_LEVEL= on page 14.

UPDATE_MULT_ROW=YES | NO
indicates whether to allow SAS to update multiple rows from a data source, such as a DBMS table. This is a LIBNAME-only option.

Default value: NO.

Some providers do not handle the following DBMS SQL statement well, and therefore update more than the current row with this statement: **`UPDATE ... WHERE CURRENT OF CURSOR`**. UPDATE_MULT_ROW= enables SAS/ACCESS to continue if multiple rows were updated.

Examples

Example 1: Specifying a LIBNAME Statement to Access OLE DB Data on on MS SQL Server In this example, the libref MYDBLIB uses the OLE DB engine to connect to an MS SQL Server database. The SAS/ACCESS-engine connection options are PROVIDER= and PROPERTIES=, and the LIBNAME options are SCHEMA= and QUALIFIER.

```
libname mydblib oledb provider=sqloledb
        properties=("User ID=shala" Password=myhrpw
                    "data source"=dept203
                    "initial catalog"=mgronly)
        schema=rfcmgrs qualifier=hrdiv;

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

Example 2: Specifying a LIBNAME Statement to Access OLE DB Data in Oracle In this example, the libref MYDBLIB uses the SAS/ACCESS engine for OLE DB to connect to an ORACLE database. Prompting is enabled, and when the dialog window opens, you supply information for the user ID, password, and data source. The SAS/ ACCESS-engine connection options are PROVIDER=, PROPERTIES=, and PROMPT=.

```
libname mydblib oledb provider=msdaora properties=("User ID"=fred
        password=freddie "data source"="v2o7223.world")
        prompt=yes preserve_tab_names=yes  preserve_col_names=yes;

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

For an example of the OLE DB Services dialog window, which prompts you for more information, see "Connecting with OLE DB Services" on page 3.

Data Set Options: OLE DB Specifics

This section describes SAS data set options that use OLE DB providers to access data sources such as DBMS tables and views. In some cases, the option is fully

described in Chapter 4, "SAS/ACCESS Data Set Options", except for some OLE DB-specific detail, such as a default value. In other cases, the entire option is OLE DB-specific, so it is described fully in this chapter.

When specified in a DATA step or SAS procedure, the following data set options can be used on a SAS data set that accesses data in a source, such as a DBMS table or view. A data set option applies only to the SAS data set on which it is specified.

The SAS/ACCESS interface to OLE DB supports all of the SAS/ACCESS data set options listed in Chapter 4, "SAS/ACCESS Data Set Options" , except for DBPROMPT=. In addition to the supported options, the following data set options are used only in the interface to OLE DB or have OLE DB–specific aspects to them:

COMMAND_TIMEOUT=

Specifies the number of seconds to wait before a command times out.

Default value: 0 (no timeout)
Alias: TIMEOUT=
Option type: LIBNAME and Data Set

Syntax

COMMAND_TIMEOUT=*number-of-seconds*

Details COMMAND_TIMEOUT= specifies the number of seconds to wait before a data source command times out.

CURSOR_TYPE=

Specifies the cursor type for read only and updatable cursors.

Default value: Not set

Alias: CURSOR=

Option type: LIBNAME and Data Set

Syntax

CURSOR_TYPE=DYNAMIC | KEYSET_DRIVEN | STATIC

Details When your data source is a relational DBMS, not all database drivers support all cursor types. An error is returned if the specified cursor type is not supported.

If CURSOR_TYPE=DYNAMIC, then the cursor reflects all of the changes that are made to the rows in a result set as you scroll around the cursor. The data values and the membership of rows in the cursor can change dynamically on each fetch. The OLE DB properties that are applied to an open row set are DBPROP_OTHERINSERT=TRUE and DBPROP_OTHERUPDATEDELETE=TRUE.

If CURSOR_TYPE=KEYSET_DRIVEN, then the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows will be reflected as you scroll around the cursor. The OLE DB properties that are applied to an open row set are DBPROP_OTHERINSERT=FALSE and DBPROP_OTHERUPDATEDELET=TRUE.

If CURSOR_TYPE=STATIC, then the cursor builds the complete result set when the cursor is opened. No changes made to the rows in the result set after the cursor is opened will be reflected in the cursor. Static cursors are read-only. The OLE DB properties that are applied to an open row set are DBPROP_OTHERINSERT=FALSE and DBPROP_OTHERUPDATEDELETE=FALSE.

By default, CURSOR_TYPE= is not set and the provider will use a default. See your provider documentation for more information. See OLE DB programmer reference documentation for details about these properties.

The CURSOR_TYPE= option can be specified with the alias CURSOR.

DBFORCE=

specifies whether to force the truncation of data during insert processing.

Default value: NO

Option type: Data set only

Syntax

DBFORCE=YES | NO

Details

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

See Also

DBKEY= in Chapter 4, "SAS/ACCESS Data Set Options"

DBINDEX=

Indicates whether or not SAS calls the data source to find index(es) on the specified DBMS table.

Default value: NO

Option type: Data set only

Syntax

DBINDEX=YES | NO | <'>*index-name*<'>

Details

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

See Also

DBKEY= in Chapter 4, "SAS/ACCESS Data Set Options"

DBNULL=

Indicates whether or not NULL is a valid value for the specified variables or columns.

Default value: YES

Option type: Data set only

Syntax

DBNULL=(*column-name-1*=YES | NO *column-name-n*=YES | NO)

Details

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

DBSASTYPE=

Specifies data type(s) to override the default SAS data type(s) during input processing of data through OLE DB.

Default value: Option is not specified.

Option type: Data set only

Syntax

DBSASTYPE=*(<column-name-1=<'>SAS-data-type<'>>*
<...<column-name-n=<'SAS-data-type<'>>>)

column-name
specifies a column name in a data source.

SAS-data-type
specifies a SAS data type, which can be one of the following: CHAR(*n*), NUMERIC, DATETIME, DATE, TIME.

Details This option is valid only when you read data into SAS through OLE DB.

By default, the SAS/ACCESS Interface to OLE DB converts each OLE DB data type to a predetermined SAS data type when processing data through OLE DB. When you need a different data type, you can use DBSASTYPE= to override the default data type selected by the SAS/ACCESS engine. Any errors are written to the SAS log. This option applies to cases in which you insert data into or append data to a data source, such as a DBMS table.

In the following example, the data stored in the DBMS FIBERSIZE column has a data type that provides more precision than what SAS could accurately support, such as DECIMAL(20). If you just used a PROC PRINT on the DBMS table, the data might be rounded or displayed as a missing value. Instead, you could use the DBSASTYPE= option to convert the column to a character field of the length 21. Because the conversion is performed before the data is brought into SAS, there is no loss of precision.

```
proc print data=mylib.specprod
   (DBSASTYPE=(fibersize='CHAR(21)');
run;
```

You can also use the DBSASTYPE= option in cases where you are appending one data source table to another table and the data types are not comparable. If the SAS data set has a variable CITY defined as CHAR(20) and the table has a column defined as DECIMAL (20), you can use DBSASTYPE= to make them match:

```
proc append base=dblib.hrdata (DBSASTYPE=(city='CHAR(20)'));
            data=saslib.personnel;
run;
```

DBSASTYPE= specifies to SAS that the CITY is defined as a character field of length 20. When a row is inserted from the SAS data set into a data source, such as a DBMS table, OLE DB performs a conversion of the character field to the DBMS data type, DECIMAL(20).

See "OLE DB Data Types" on page 41 for details about the default data types for OLE DB.

DBTYPE=

Specifies data types(s) to override the default OLE DB data type(s) when SAS outputs data to DBMS tables through OLE DB.

Default value: DBTYPE_STR(*size*) or DBTYPE_R8

Option type: Data set only

Syntax

DBTYPE=*(column-name-1=data-type <...> <column-name-n=data-type>)*

Details

The default type for SAS character variables is DBTYPE_STR(*size*), where the size is derived from the length of the SAS variable length. The default for SAS numeric variables is DBTYPE_R8 unless you specify a format such as *m.n*, in which case the type becomes DBTYPE_NUMERIC. For more information on SAS and OLE DB types, see “OLE DB Data Types” on page 41. To see a full description of this data set option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

QUALIFIER=

Specifies the qualifier to use when reading a data source, such as DBMS tables and views.

Default value: default schema name

Option type: LIBNAME and Data set

Syntax

QUALIFIER=<*qualifier-name*>

Details QUALIFIER= is optional. If it is omitted, you use the default qualifier name, if any, for the data source. QUALIFIER= can be used for any data source, such as a DBMS object, that allows three-part identifier names: *qualifier.schema.object*. For example, in the following SAS/ACCESS LIBNAME statement, any reference to SAS in the DBLIB.EMP table would be sent to the DBMS as **`rfcdept.hrdiv.emp`**.

```
libname dblib oledb provider=sqloledb
 properties=('User ID'=suzanne Password=mypw3
             "data source"=t1007
             "initial catalog"=mgr1)
schema=hrdiv qualifier=rfcdept;
```

READ_ISOLATION_LEVEL=

Defines the degree of isolation of the current application process from other concurrently running application processes.

Default value: Provider specifies

Alias: RIL=

Option type: LIBNAME and Data set

Syntax

READ_ISOLATION_LEVEL=S | RR | RC | RU

S = Serializable

RR = Repeatable Read

RC = Read Committed

RU = Read Uncommitted

Details

The degree of isolation identifies

- the degree to which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree to which update activity of other concurrently executing application processes can affect the current application.

The READ_ISOLATION_LEVEL= option applies only when reading a DBMS table or view. By default, this option is not set. The provider sets the default value. If READ_LOCK_TYPE= is not set to ROW, then READ_ISOLATION_LEVEL= is ignored .

OLE DB supports five isolation levels. The isolation levels are defined in terms of several possible occurrences:

- Dirty read — A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it will be able to see changes 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 may 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 will see a row that did not previously exist, a phantom.

The isolation levels for READ_ISOLATION_LEVEL= include the following:

- Serializable (S)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - does not allow phantom reads
- Repeatable Read (RR)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - allows phantom reads
- Read Committed (RC)
  - does not allow dirty reads
  - allows nonrepeatable reads
  - allows phantom reads
- Read Uncommitted (RU)
  - allows dirty reads
  - allows nonrepeatable reads
  - allows phantom reads

READ_LOCK_TYPE=

Specifies how a table is locked during a READ operation.

Default value: ROW

Option type: LIBNAME and Data set

Syntax

READ_LOCK_TYPE=ROW

Details

The value ROW specifies that a row or set of rows will be locked. SAS/ACCESS software uses the READ_ISOLATION_LEVEL= option to determine which rows will be locked. Currently, the only valid locking that is allowed is through the READ_ISOLATION_LEVEL= option; therefore, READ_LOCK_TYPE= only allows a value of ROW.

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

ROWSET_SIZE=

Specifies the number of rows to use when reading data from the data source.

Default value: 1

Alias: ROWSET=

Option type: LIBNAME and Data set

Syntax

ROWSET_SIZE=*number-of-rows*

Details

By default, ROWSET_SIZE=1 so that only one row is retrieved at a time. The higher the value for ROWSET_SIZE=, the more rows that the SAS/ACCESS engine for OLE DB retrieves in one fetch operation. This option reduces the amount of I/O that is used and can help improve performance. However, because SAS software stores the rows in memory, higher values for ROWSET_SIZE= use more memory. In addition, if too many rows are selected at once, then the rows that are returned to the SAS application might be out of date.

SASDATEFMT=

Changes the SAS date or datetime format of a data source column.

Default value: Not set

Option type: Data set only

Syntax

SASDATEFMT=*(date-column="SAS-date-format" ...)*

Details

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

Example: Using SASDATEFMT to convert a date column to a SAS date format In this example, SAS changes the format of the HIRED column to the SAS DATE9. format for reading the data in SAS.

```
proc print data=mydblib.payroll
   (sasdatefmt=(hired='DATE9.'));
run;
```

See Also

SASDATEFMT= in Chapter 4, "SAS/ACCESS Data Set Options"

SASDATEINFMT=

Changes the SAS date informat of a data source column.

Default value: None

Option type: Data set only

Syntax

SASDATEINFMT=*(date-column="SAS-date-format"...)*

Details

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

Example: Using SASDATEINFMT to convert SAS dates into DBMS dates In this example, SAS changes the HIRED column from the default OLE DB data type to the DATE9. format before appending the data to the DBMS table.

```
proc append base=air.payroll
      data=mydblib.payroll2
        (sasdateinfmt=(hired='DATE9.'));
run;
```

See Also

SASDATEFMT= in Chapter 4, "SAS/ACCESS Data Set Options"

SCHEMA=

Enables you to read a data source, such as a DBMS table and view, in the specified schema.

Default value: None

Alias: OWNER=

Option type: LIBNAME and Data set

Syntax

SCHEMA=*schema-name*

Details

A schema is a logical classification of objects accessed by a data source, where the source is a relational database. For this option to work, you must have READ privileges

to the schema that is specified. SCHEMA= is optional, and if omitted, you connect to the default schema.

Example: Accessing a table using SCHEMA=

In this example, SAS sends any reference to **mydblib.emp** as **dbitest.emp.**

```
libname mydblib oledb provider=sqloledb properties=('User ID'=sue
        Password=mypw3) qualifier=hrdiv;
proc print data=mydblib.employees (schema=mitori);
run;
```

UPDATE_ISOLATION_LEVEL=

Defines the degree of isolation of the current application process from other concurrently running application processes

Default value: Not set

Alias: UIL=

Option type: LIBNAME and Data set

Syntax

UPDATE_ISOLATION_LEVEL=S | RC | RR

S = Serializable

RC = Read Committed

RR = Repeatable Read

Details

The degree of isolation identifies

- the degree to which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree to which update activity of other concurrently executing application processes can affect the current application.

By default, UPDATE_ISOLATION_LEVEL= in not set. The provider sets the default value. If UPDATE_LOCK_TYPE= is not set to ROW, then UPDATE_ISOLATION_LEVEL= is ignored.

OLE DB supports four isolation levels. The isolation levels are defined in terms of several possible occurrences:

- Dirty read — A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it will be able to 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 may 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 will see a row that did not previously exist, a phantom.

The isolation levels for UPDATE_ISOLATION_LEVEL= include the following:

- Serializable (S)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - does not allow phantom reads
- Repeatable Read (RR)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - allows phantom reads
- Read Committed (RC)
  - does not allow dirty reads
  - allows nonrepeatable reads
  - allows phantom reads

UPDATE_LOCK_TYPE=

Specifies how an OLE DB table is locked during an UPDATE operation

Default value: ROW

Option type: LIBNAME and Data set

Syntax

UPDATE_LOCK_TYPE=ROW

Details

The UPDATE_ISOLATION_LEVEL= option applies only when updating a DBMS table or view. The value ROW specifies that a row or set of rows will be locked. To determine which rows will be locked, SAS/ACCESS software uses the UPDATE_ISOLATION_LEVEL= option.

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

See Also

READ_LOCK_TYPE=“READ_LOCK_TYPE=” on page 22

SQL Procedure Pass-Through Facility: OLE DB Specifics

The SQL Procedure Pass-Through facility consists of three PROC SQL statements and one component. For details about the OLE DB-specific information, see the “CONNECT Statement” on page 27 and the “CONNECTION TO Component” on page 32 component. For a complete description of the SQL Procedure Pass-Through facility, see Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software" .

CONNECT Statement

Establishes a connection with the data source.

Syntax

CONNECT TO OLEDB <AS *alias* > <(<*OLE-DB-connection-arguments*>)>;

Arguments

You use the following arguments with the CONNECT statement:

alias
specifies an optional alias that has 1 to 32 characters. If you specify an alias, the keyword AS must appear before the alias.

(OLE DB-connection-arguments)
specifies the data source-specific arguments that PROC SQL needs in order to connect to the data source, such as a relational DBMS. These arguments must be enclosed in parentheses. For some data sources, these arguments have default values and, therefore, are optional. The arguments for OLE DB are described in the following sections.

OLE DB Connection Arguments 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 is required when connecting to OLE DB providers by way of the SQL Pass-Through Facility.

Note: Not all of these engine-connection options are supported by all OLE DB providers. Refer to your vendor-supplied documentation for more information. △

The arguments for the data-source connection information arguments can be quoted by using either single or double quotes. Some values may include embedded spaces, semicolons, or quotes and, therefore, must be quoted.

The OLE DB connection arguments are

AUTOCOMMIT=YES | NO
indicates whether or not updates are committed immediately to the data source as soon as they are submitted.

Default value: YES.

If AUTOCOMMIT=YES, no rollback is possible. This is the default for the SQL Procedure Pass-Through Facility and read-only connections.

If AUTOCOMMIT=NO, the SAS/ACCESS engine performs the commit automatically when it reaches the end of the file.

COMPLETE=YES | NO
specifies whether the SAS/ACCESS engine for OLE DB tries to connect to the data source with the properties that you specified in PROPERTIES=. If COMPLETE=YES, the engine tries to connect. If enough information is specified for a successful connection, then the connection is made without any prompting for more information.

Default value: NO.

If you specify COMPLETE= NO or do not have enough information to connect, you are prompted for the connection options with a dialog window.

COMPLETE= is used only when you use connect directly to OLE DB using the PROVIDER= and OLEDB_SERVICES=NO options in a SAS/ACCESS LIBNAME statement.

See also these arguments: PROMPT= on page 30, PROPERTIES= on page 30, PROVIDER= on page 30, OLEDB_SERVICES= on page 29.

INIT_STRING="<*initilization-string*>"
using OLE DB Services, INIT_STRING= specifies an initialization string when connecting to a data source. After you are prompted to supply information to connect to your data source, the SAS/ACCESS engine for OLE DB returns the complete initialization string to the macro variable, SYSDBMSG. You can then re-use the initialization string to connect to the same provider and data source.

In this example, you submit the basic connection information so that you will be prompted for the rest of the information. Using OLE DB Services is the default value, so you can omit the OLEDB_SERVICES= option.

```
libname mydblib oledb;
```

Through dialog windows, OLE DB Services prompts you for the provider and properties' values. The advantage of being prompted is that you do not need to know any special syntax to set the properties'. Prompting also enables you to set more options than you might when connecting directly to the provider (and not using OLE DB Services).

After connecting to the data source, the SAS/ACCESS engine for OLE DB returns the initialization string to the SYSDBMSG macro variable. To write this

string to the SAS log *immediately* after connecting to the data source, submit the following:

```
%put &SYSDBMSG;

OLEDB:  Provider=SQLOLEDB;Password=dbmgr1;Persist Security Info=True;
User ID=rachel;Initial Catalog=users;Data Source=DBPC6;
```

The SYSDBMSG information mirrors all of the options that you chose during your prompted connection. Notice that the initialization string is prefixed with **OLEDB:**. When you store the string for later use, delete this prefix and any initial spaces before the first listed option.

By storing the initialization string, you can re-use it in the INIT_STRING= option to make automated connections or to specify this option in batch jobs:

```
init_string="Provider=SQLOLEDB;Password=dbmgr1;
Persist Security Info=True;User ID=rachel;
Initial Catalog=users;Data Source=DBPC6";
```

Using INIT_STRING= enables you to bypass the prompting window but still gives you the advantages of the OLE DB Services, such as performance optimizations.

Specifying INIT_STRING= overrides any values that were set with the SAS/ACCESS connection options PROVIDER= on page 30 and PROPERTIES= on page 30.

By default, the INIT_STRING= option is not set. However, if you specify the option with a null argument, (**INIT_STRING=""**), OLE DB connects to ODBC with a default set of properties. See the Microsoft OLE DB documentation for more information about these defaults.

Alias: INIT=.

OLEDB_SERVICES=YES | NO
determines whether the SAS/ACCESS engine for OLE DB uses OLE DB Services. OLEDB_SERVICES=YES causes the engine to use OLE DB Services, and OLEDB_SERVICES=NO causes the engine to use the provider to connect to the data source.

The default value is YES. Generally, OLE DB Services is easier to use and more consistent. When OLEDB_SERVICES=YES and a successful prompted connection is made, the complete connection string is returned in the SYSDBMSG macro variable.

OLEDB_SERVICES= interacts with other connection options. If you have set PROMPT=YES, OLEDB_SERVICES=YES enables you to set more options than you would be able to set by being prompted by the provider's dialog window. If OLEDB_SERVICES=NO, you must specify PROVIDER= first in order for the provider's prompt dialogs to be used. If PROVIDER= is omitted, the SAS/ACCESS engine uses OLE DB Services, regardless of how the OLEDB_SERVICES= option is set.

If the BCP=YES option is set for MS SQL Server data, then OLEDB_SERVICES=NO. OLEDB_SERVICES= also interacts with the PROMPT=, REQUIRED= , and COMPLETE= arguments.

See these arguments for more information: COMPLETE= on page 28, PROMPT= on page 30, PROVIDER= on page 30, REQUIRED= on page 31.

PROMPT=YES | NO
enables you to be prompted for connection information to access the data source. The kind of prompting that you receive depends on how you set the PROVIDER= and OLEDB_SERVICES= options.

If a provider name is specified *and* OLEDB_SERVICES= NO, the OLE DB provider displays a dialog window that contains the connection information and property attributes. If the provider name is omitted *or* OLEDB_SERVICES=YES, the OLE DB Services displays a dialog window that enables you to select a provider and to specify connection information and property attributes. The dialog window for OLE DB Services is generally preferred over the provider's dialog window because the OLE DB Services window enables you to set options more easily.

If PROMPT=YES, properties that were set with PROPERTIES= will be displayed in the dialog window. This applies both to the provider dialog window and to the OLE DB Services dialog window. You can edit any field before you connect to the data source.

If the provider name is omitted, the SAS/ACCESS engine for OLE DB tries to prompt you for the connection information by using the OLE DB Services dialog window. This applies even if PROMPT=NO and OLEDB_SERVICES= NO. If the provider name is omitted in batch mode, the connection fails.

If you are unsure what to specify for various provider properties, use the PROMPT= option to guide you through the connection process.

See the following arguments for more information: PROVIDER= on page 30, OLEDB_SERVICES= on page 29, PROPERTIES= on page 30.

PROPERTIES=*(<">property-name-1<">=<">property-value-1<"> . . . <">property-name-n <">=<">property-value-n<">)*
specifies provider properties that enable you to connect to a data source and to define the attributes of that connection. Each property name is assigned a value using an equal sign (=). If the property name or value contains embedded spaces or special characters, enclose the name in double quotes. Separated multiple pairs with a space. PROPERTIES= is optional in OLE DB.

In this example, you specify a user ID and password to connect to a Microsoft SQL Server data source, you would enter:

```
libname mydblib oledb provider=sqloledb
                properties=("User ID"=shala
                Password="mypw@hr");
```

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

Aliases: PROPS= and PROP=.

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

There is no restriction on the length of the name. Put names with non-standard SAS characters (such as spaces, colons, @ signs) in single or double quotations marks.

If you omit this option, you are prompted for the provider name. It is recommended that, if possible, you use the dialog prompts to connect to your data source. The prompts enable you to use an interactive interface to enter the name of the provider, properties, and connection options.

Alias: PROV=.

PROVIDER_STRING=<">*provider-name*<">
passes additional provider-specific connection information to the provider. The provider-string is enclosed in quotation marks if it contains blank spaces or special characters. The PROVIDER_STRING= option works whether you are connecting directly to the provider or are using OLE DB Services. Microsoft uses this provider-string for its Jet provider in order to determine the type of data source to which it connects. MS Jet currently accepts the following providers; this list is not all-inclusive and is subject to change by Microsoft:

dBase III, IV, 5.0
Excel 3.0, 4.0, 5.0, 8.0
Exchange 4.0
HTML Export, HTML Import
Jet 2.*x*, Jet 3.*x*
Lotus WJ2, WJ3
Lotus WK1, WK2, WK3, WK4
Outlook 9.0
Paradox 3.*x*, Paradox 4.*x*, Paradox 5.*x*, Paradox 7.*x*
Text

Providers other than MS Jet accept other provider-strings, and the values that MS Jet accepts are subject to change. For example, to connect to an Excel 8.0 spreadsheet using the Microsoft Jet provider, you could issue the following LIBNAME statement:

```
libname y2kbudget oledb provider="Microsoft.Jet.OLEDB.4.0"
  properties=('data source'='d:\excel80\Y2Kbudgetworksheet.xls')
  provider_string="Excel 8.0";
```

In this example's LIBNAME statement, you use the Microsoft Jet 4.0 provider to access the spreadsheet **`Y2Kbudgetworksheet.xls`**. Notice that you must specify the provider-string **`"Excel 8.0"`** so that MS Jet recognizes that the file is an Excel 8.0 worksheet.

REQUIRED=YES | NO
indicates whether you specify connection options for your provider.

If you specify REQUIRED= YES, the SAS/ACCESS engine for OLE DB tries to connect to the data source by using the properties that were specified in PROPERTIES=. If you specify enough information to make a connection, then the connection is made without prompting. Otherwise, a dialog window is displayed to prompt you for the connection options. Options in the dialog window that are not related to the connection are disabled.

REQUIRED= is used only when you connect directly to OLE DB using the PROVIDER= and OLEDB_SERVICES=NO options in a SAS/ACCESS LIBNAME statement. The default value is REQUIRED=NO.

For more information, see OLEDB_SERVICES=, PROPERTIES= on page 30, PROVIDER= on page 30.

Example: An MS SQL Server CONNECT statement In this example, you use 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 sqlservr as finance
  (properties=('data source'='Microsoft SQL Server Database'
     'User ID'=isabella password=tester1
   provider=sqloledb));

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

In this example, you are prompted for more information because the PROC SQL CONNECT statement has omitted the provider name and properties. See "Connecting with OLE DB Services" on page 3 for a sample prompt window. This example also uses OLE DB Services to connect to OLE DB.

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

CONNECT Example This example uses OLE DB to connect to a provider that is configured under the data source name **`User's Data`** using the alias USER1. Note that the data source name(s) can contain quotes and spaces.

```
proc sql;
   connect to oledb as user1
   (properties=("data source"="User's Data" "User id"=adi
                password=ghana));
```

CONNECTION TO Component

Retrieves and uses data from a data source in a PROC SQL query or view.

Optional component

Syntax

CONNECTION TO OLEDB | *alias (DBMS-SQL-query)*

Arguments

alias
specifies an alias, if one was defined in the CONNECT statement.

(DBMS-SQL-query)
specifies the query that you are sending to the data source, such as a relational DBMS. Data sources other that relational databases can be specified in the SQL Procedure Pass-Through Facility, but they must have SQL capabilities. The query can use any DBMS-specific SQL statement or syntax that is valid for the DBMS.

However, the query cannot contain a semicolon because to the SAS System, a semicolon represents the end of a statement.

You must specify a *DBMS-SQL-query* argument in the CONNECTION TO component, and the query must be enclosed in parentheses. The query is passed to the DBMS exactly as you type it; therefore, if your DBMS is case sensitive, you must either use the correct case for names or you must quote them. Quoted character strings are limited to 32Kb characters. Or for some DBMSs, the *DBMS-SQL-query* argument can be a DBMS-specific SQL EXECUTE statement that executes a DBMS stored procedure. However, if the stored procedure contains more than one query, only the first query is processed.

The CONNECTION TO component specifies the data source connection that you want to use or that you want to establish (if you have omitted the CONNECT statement). CONNECTION TO then enables you to retrieve data from the data source directly through a PROC SQL query.

You use the CONNECTION TO component in the FROM clause of a PROC SQL SELECT statement:

PROC SQL;

SELECT *column-list*

FROM CONNECTION TO *dbms-name (DBMS-SQL-query)*
other-optional-PROC-SQL-clauses;

CONNECTION TO can be used in any FROM clause, including those that are in nested queries (that is, subqueries).

You can store a Pass-Through query in a PROC SQL view and then use that view in SAS programs. When you create a PROC SQL view, any options that you specify in the corresponding CONNECT statement are stored too. Thus, when the PROC SQL view is used in a SAS program, the SAS System can establish the appropriate connection to the data source.

On many DBMSs, you can issue a CONNECTION TO component in a PROC SQL SELECT statement directly without first connecting to a DBMS . If you omit the CONNECT statement, an implicit connection is performed when the first PROC SQL SELECT statement that contains a CONNECTION TO component is passed to the DBMS. Default values are used for all connection arguments.

Because DBMSs and the SAS System have different naming conventions, some DBMS column names might be truncated when you retrieve DBMS data through the CONNECTION TO component. Default SAS variable names follow these rules:

- If the column name is longer than thirty-two characters, the SAS System uses only the first thirty-two characters. If truncating results in duplicate names, sequential numbers (starting with zero) are appended to the ends of the names.
- If the column name contains characters that are invalid SAS names (such as national characters), the SAS System replaces the invalid characters with underscores (_). For example, the column name **`func$`** becomes the SAS variable name **`func_`**.

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. The general format of the special queries is:

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 the *"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 "Owner" might be a user ID, a database name, or a library. All arguments are optional. If you specify some but not all parameters within an argument, use a comma to indicate the omitted parameters. 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 example“Examples” on page 36.

OLEDB::PROVIDER_TYPES(<*"Data Type", "Best-Match"*>)
returns information on 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

In this example, you retrieve a rowset that displays all of the tables that are accessed by the schema HRDEPT:

```
proc sql;
  connect to oledb("User ID"=dbajorge Password=dbajorge99
     "Data Source"="oracle8_loc");
  select * from connection to oledb
       (OLEDB::TABLES(,"HRDEPT"));
```

This next example uses the special query OLEDB::PROVIDER_INFO() to produce the output that follows it:

```
proc sql;
  connect to oledb("User ID"=dbajorge Password=dbajorge99
     "Data Source"="oracle8_loc");
```

```
select * from connection to oledb
     (OLEDB::PROVIDER_INFO());
```

Output 1.1 Provider and Properties Output

PROVIDER_NAME	PROVIDER_DESCRIPTION	PROVIDER_PROPERTIES
MSDAORA	Microsoft OLE DB Provider for Oracle	Password;User ID;Data Source;Window Handle;Locale Identifier;OLE DB Services; Prompt; Extended Properties;
SampProv	Microsoft OLE DB Sample Provider	Data Source;Window Handle; 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'=Smith);
```

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 SQL Procedure Pass-Through Facility to access the data directly, or create a PROC SQL view of the data. Note that you must pass an MDX statement which specifies a two-axis "flattened" data set. Attempting to return a data set with more than two axes will result in an error. Refer to the Microsoft Data Access Components Software Developer's Kit for details on 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 SQL Procedure Pass-Through Facility

The main difference between normal OLE DB access using SQL Pass-Through and this 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:

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 on each returned column. During the process of “flattening” multidimensional data into a two-axis data set, 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 SASNAMEs 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.

Syntax

This facility uses the following general syntax. For more information on SQL Pass-Through syntax, see .

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)*>
;

MDX::
identifies the following statement as an MDX statement that requests data from the multidimensional data. The MDX statement is passed through to the provider, and the resulting “flattened” data set is returned to SAS software.

MDX_DESCRIBE::
identifies a request for detailed information about the column names for the data set that would be returned by the MDX statement. The returned data set contains two varaibles:

- SASNAME, containing the SAS names for the columns that would be returned by the MDX statements
- MDX_UNIQUE_NAME, containing the fully-described OLE DB for OLAP column identifier

Examples

The following code uses SQL Pass-Through 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 on MDX systax.

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 sytax, 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 PROC 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.

OLE DB Naming Conventions

Because OLE DB is not a database but rather is an application programming interface (API), data source names for files, tables, and columns are determined at run time, as described here. 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 file, table, or column names are longer than 32 characters, they will be truncated to 32 characters. For more information, see Chapter 2, "SAS Names and Support for DBMS Names" .

PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= are two SAS/ACCESS LIBNAME options that specify whether to preserve blank spaces, special characters, and mixed case in file, table, or column names. By setting these options to YES, SAS preserves the case-sensitivity of the names. If PRESERVE_TAB_NAMES=NO and PRESERVE_COL_NAMES=NO, then file, table, and column names that are read from the data source are converted to SAS names by using the SAS-name normalization rules.

This example specifies a Microsoft SQL Server provider that interacts with OLE DB. SQL Server is generally case-sensitive and, as a provider to OLE DB, it takes the default value YES for the preserve-name options. Therefore, this example would produce a SQL Server table named **staffids** with a column named **ID Num**.

```
options validvarname=any;

libname mydblib oledb provider=sqloledb properties=('UserID'=DIETER
        password=FRUEHAUF "data source"="HR@00123");

proc sql dquote=ansi;
create table staffids as
  select IDNUM as 'ID Num'n
   from mydblib.STAFF;
quit;

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

If the data source were ORACLE, which is not case sensitive and which has a default value of NO for the preserve-name options, the example would produce an ORACLE table named **STAFFIDS** with a column named **ID_NUM**. The column names would be normalized to uppercase and an underscore would be substituted for blank spaces or characters that are not valid in SAS names.

For more information about the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options, see "SAS/ACCESS LIBNAME Options" on page 9.

OLE DB Data Types

A data source's columns in a table each have 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 are stored.

Table 1.2 on page 41 shows all of the data types and default SAS formats that are supported by the SAS/ACCESS engine for 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 1.2 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.
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.

OLE DB Data Type	Default SAS Format
DBTYPE_DBTIME	TIME8.
DBTYPE_TIMESTAMP and DBTYPE_DATE	DATETIME*m.n*, where *m* depends on precision and *n*depends on scale

Table 1.3 on page 42 shows the default data types that the SAS/ACCESS engine for OLE DB uses when creating DBMS tables.

Table 1.3 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_DBIME
datetime formats	DBTYPE_DBTIMESTAMP

The SAS/ACCESS engine for OLE DB allows non-default data types to be specified with the DBTYPE= data set option. See “DBTYPE=” on page 20 for more information about this data set option.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Software for Relational Databases: Reference, Version 8 (OLE DB Chapter), Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS/ACCESS® Software for Relational Databases: Reference, Version 8 (OLE DB Chapter)

ISBN 1-58025-552-3

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999