Configuring and Using ODBC with SAS/ACCESS® Software
Forrest Boozer - SAS Institute Inc.

ABSTRACT

ODBC has grown explosively on the Windows platform because it allows application developers to develop a single DBMS interface that maps (via ODBC) to almost every DBMS and file format available. Yet, even with its wide acceptance, users find ODBC difficult to understand and discuss. This paper explains the architecture of ODBC and common terms. Based on those terms, the SAS/ACCESS interface to ODBC will be presented and the effect of its various options on the ODBC manager/driver will be detailed. Finally, configuring data sources using the ODBC administrator will be discussed with examples of several data source configurations.

ODBC OVERVIEW

Open Database Connectivity, ODBC, is a specification for a common API (Application Programming Interface) for connecting to different databases. Client programs, which support the ODBC specification, can access any database which provides an ODBC driver. Support for a common API also allows programs to access diverse databases without any code modifications. For example, a program could be developed using a Microsoft Access database for data storage, and deployed using Oracle® on a remote server simply by specifying a different ODBC data source with no change to the underlying program code.

ODBC is tightly coupled to SQL (Structured Query Language) as both the syntax for making requests of a database, as well as the model for presentation of the data - all data must be presented as if it were table oriented. This does not mean that you must learn SQL in order to benefit from ODBC. Many popular programs which access data through ODBC include a front end to present the data in a graphical interface, generating SQL queries in the background. The SAS® System provides this capability in the Query window.

ODBC generalizes the services provided by databases in much the same way that printer drivers generalize the services provided by printers. ODBC conformant applications do not require any customization to access new databases, provided they support an adequate level of ODBC capability. To go further with the printer example, if a program requires high resolution and color, a consistent interface will not make an inadequate printer work, but will allow the application to tell the difference. To take the analogy to ODBC, some drivers will be read only, some will not provide index information or directory type functions, while most tend to be highly functional. The ODBC specification allows programs to determine the available functionality and make use of what is provided.

ODBC COMPONENTS

ODBC functionality is provided by five components: the client interface application, the ODBC driver manager, the ODBC driver administrator, the ODBC driver, and the data source.

Client Interface Application - is an application program which uses the ODBC API to access data. The SAS System, using the SAS/ACCESS interface to ODBC pass-through engine, is a client application.

ODBC Driver Manager - is the DLL (Dynamic Link Library) to which a client application (such as the SAS System) is linked. It loads the appropriate driver requested by the client application for a connection and does some system level management, but primarily passes all calls through to the ODBC driver.

ODBC Administrator - is used to install drivers and create/configure data sources.

ODBC Driver - is the DLL which actually provides the interface to a specific data source. Drivers are provided by the database vendor or third party vendors, like Microsoft, Visigenic, and Intersolv.

Data Source - consists of a specific instance of the data itself or any DBMS, networking or operating system software required to access the data.

Figure 1: Relationship of ODBC Components

The figure above depicts the relationships between the various ODBC components. An ODBC client application makes calls only to the ODBC Driver Manager which is responsible for routing the call to the appropriate ODBC driver. The driver interfaces to the Data Source to service the ODBC call originated by the client application. The ODBC manager is not involved in the run-time call sequence of a client application, but is used for configuring ODBC Data Source Names prior to their use in client applications.

On Windows platforms, Microsoft allows ODBC driver vendors to freely ship the Microsoft ODBC Driver Manager and Administrator with third party drivers. On non-Windows systems, the ODBC Driver Manager and optionally the ODBC Administrator are provided by the driver vendors themselves. Two vendors of note are Visigenic and Intersolv both of which develop drivers for UNIX and other platforms including OS/2® and Macintosh. Visigenic has a license from Microsoft to port the Windows based ODBC SDK to non-Windows platforms. This license allows them to port Microsoft's Driver Manager and Administrator to other platforms. On the other hand, Intersolv has written its own ODBC driver manager from scratch for non-Windows platforms.

ODBC LEVELS OF CONFORMANCE

The actual interface between the ODBC client application and the ODBC driver manager is provided in two mechanisms. The first is...
the actual ODBC function API which defines the function calls that
may be made by the client application. The second is the ODBC
Structured Query Language, statements of which are passed to the
ODBC Driver via ODBC API functions.

The ODBC specification supports a very rich functionality when both
the ODBC API and SQL Grammar are fully implemented.
Depending on the data source and the specific driver, the full ODBC
specification may not be implemented for a particular driver. To help
users determine whether a specific driver will meet their needs,
Microsoft has established guidelines that a driver must meet in order
to claim a certain level of ODBC conformance.

Microsoft established two sets of guidelines for ODBC conformance,
one for the ODBC function API, and the other for ODBC SQL
grammar. Although the specification indicates that drivers must
meet all of the requirements for a given level in order to claim that
level of conformance, drivers generally claim conformance for levels
where they support the majority of requirements, and then simply
note exceptions in their documentation.

(Information concerning ODBC conformance was obtained from the
Programmer's Reference: Microsoft Open Database Connectivity
Software Development Kit, Version 2.0 from Microsoft Corporation.)

ODBC API Conformance

For the ODBC API, drivers may be categorized into three levels of
compliance: CORE, Level 1, and Level 2 (or low, medium and high
levels of functionality, respectively). The following can be expected
of the different conformance levels. This is not the complete list;
there may be additional requirements for each level.

CORE API

- Multiple statements can be executed on a single connection.
- Statements can be prepared and described, data can be
  retrieved from a result set.
- Commit and rollback are supported.
- Error information is returned.

Level 1 API

- Supports full CORE API functionality.
- Dynamically connects to data sources using driver supplied
dialog boxes.
- Can set and retrieve statement and connection options.
- Data source descriptive information (tables, columns,
  statistics) can be retrieved.
- Driver descriptive information (supported data types, scalar
  functions, and ODBC functions) can be retrieved.

Level 2 API

- Core and Level 1 API functionality
- Lists available connections and available data sources.
- Supports implements scrollable cursors.
- A higher level of data source descriptive information (privileges,
  keys, and procedures) can be retrieved.

ODBC SQL Conformance

The SQL grammar supported also has three levels of compliance:
minimum, core and extended (again, low, medium and high
respectively). For SQL grammar conformance, the following can be
expected. Please note, drivers have the ability to support not only
the official ODBC SQL grammar, but they may extend the grammar
by allowing the data source's native SQL to pass through the driver
to the data source.

Minimum SQL Grammar

- Data definition statements CREATE TABLE and DROP
  TABLE are supported.
- Data manipulation statements for simple SELECT, INSERT,
  UPDATE, and DELETE are supported. These statements
  support a WHERE clause and simple expressions.
- Only the CHAR data type is required.

Core SQL Grammar

- Minimum SQL grammar.
- Data definition statements such as ALTER TABLE, CREATE
  INDEX, CREATE VIEW, GRANT, and their dropping
  counterparts are supported.
- The full SELECT statement syntax is supported.
- Subqueries and set functions such as SUM and MIN are
  supported in expressions.
- Additional data types: VARCHAR, DECIMAL, NUMERIC,
  SMALLINT, INTEGER, REAL, FLOAT, DOUBLE
  PRECISION are implemented.

Extended SQL Grammar

- Minimum and Core SQL grammar.
- Complex data manipulation such as outer joins and unions,
  positioned UPDATE and DELETE, SELECT FOR UPDATE,
  and unions are supported.
- Scalar functions, date, time and timestamp literals may be used
  in expressions.
- Additional data types of: LONG VARCHAR, BIT, TINYINT,
  BIGINT, BINARY, VARBINARY, LONG VARBINARY, DATE,
  TIME, TIMESTAMP are supported.
- Procedure calls may be made within SQL statements.

TYPES OF ODBC DRIVERS

There are two basic types of ODBC drivers, single tiered and multi-
tiered.

![Single Tier Driver](Figure 2: Single Tier ODBC Driver)

A single tier driver is a driver which operates on data which resides
on the same machine or network using direct reads and writes to the
data file. Typical examples of single tier drivers are drivers for .DBF
files, Excel .XLS files and text files.

![Multi-Tier Driver](Figure 3: Multi-Tier ODBC Driver)

A multi-tiered driver does not directly manipulate database data. It
generally makes calls to a DBMS-provided API that allows the driver
to communicate directly with the DBMS server as a client
application. The API provided by the DBMS may or may not be SQL
Based. In either case, the ODBC driver is responsible for making the appropriate calls into the DBMS to service the ODBC API calls. Examples of multi-tiered ODBC drivers are those for DB2, SQL Server, Oracle, and AS/400.

Configuration of multi-tiered ODBC data sources also requires the configuration of the Data Access API (pictured above). This can be a simple or complex task depending on the data source you are trying to access and its location.

**USING THE SAS/ACCESS INTERFACE TO ODBC**

The figure 4 shows the relationships between the SAS System and ODBC components. Within the SAS System, PROC SQL and the SAS/ACCESS interface to ODBC are used to access ODBC data sources. PROC SQL provides the framework within the SAS System for passing SQL statements to relational databases. The SAS/ACCESS interface to ODBC communicates with the ODBC driver manager via the ODBC API and ODBC SQL grammar as discussed earlier. These API calls are then routed by the driver manager to the appropriate ODBC driver and ultimately the desired data source.

![Figure 4: SAS System and ODBC Components](image)

The SAS/ACCESS interface to ODBC is an I/O engine that is accessed through the PROC SQL Pass-Through facility. There are four basic statements that are used in PROC SQL to access ODBC Data Sources:

- **CONNECT TO ODBC** (connect options);
- **SELECT * FROM CONNECTION TO ODBC (ODBC SQL select stmt);**
- **EXECUTE** (non-select ODBC SQL stmt) **BY ODBC;**
- **DISCONNECT FROM ODBC;**

The **CONNECT** statement establishes a connection with the ODBC data source. This statement is required and must precede all other statements which will use ODBC data. This statement is directly impacted by the configuration of ODBC data sources. There are many options that may be used on the **CONNECT** statement; these will be discussed later.

The **SELECT FROM CONNECTION TO ...** statement follows a **CONNECT** statement, and is used to send ODBC SQL queries to a data source and retrieve the results. Queries are usually in the form of a SELECT statement but may also reference a procedure that returns results.

The **EXECUTE** statement is used to send NON-Query ODBC SQL statements to a data source. Examples of such statements would be those for creating database objects (tables, views, indexes) or inserting/updating data in existing objects.

Note that the ODBC SQL statements enclosed in the parenthesis for the **SELECT** and **EXECUTE** statements are ODBC SQL specific. They are not SAS SQL statements. These statements are passed directly to the ODBC driver, and ultimately to the DBMS being accessed. So, when entering these statements, use the DBMS table names and column names, not the derived SAS names.

The **DISCONNECT** statement terminates an existing connection to a data source. Use of the **DISCONNECT** statement is not required unless you need to re-establish a connection to another data source or free resources occupied by unused connections during the execution of PROC SQL. When PROC SQL terminates, all existing connections are implicitly disconnected.

**Examples of Using PROC SQL Pass-Through With ODBC**

PROC SQL supports three methods for returning data to the SAS System. By default PROC SQL statements generate a listing of the requested data. Output from PROC SQL statements can also be used to create SAS data sets. Finally, PROC SQL can be used to define SAS mappings, or views, of relational data. The following examples illustrate how PROC SQL can be used to access ODBC data sources.

Suppose you have defined a data source called "SALES" for your ORACLE corporate sales database. To get a listing of all the ACCOUNTS in North Carolina you could use the following SAS statements.

```sas
PROC SQL;
CONNECT TO ODBC (REQUIRED="DSN=SALES");
SELECT * FROM ACCOUNTS WHERE STATE = "NC"
WHERE STATE = "NC"
SELECT * FROM ACCOUNTS WHERE STATE = "NC"
```

To use this same information as input for SAS procedures, you can create a SAS dataset. The following statements will create a SAS dataset, MYLIB.NCACCTS that is a snapshot of the data from the Oracle table.

```sas
PROC SQL;
CONNECT TO ODBC (REQUIRED="DSN=SALES");
CREATE TABLE MYLIB.NCACCTS AS
SELECT * FROM ACCOUNTS WHERE STATE = "NC"
```

Alternatively, you can create a PROC SQL view of this same data for later use in SAS programs. Views do not contain actual data, but rather define mappings to data that resides in the DBMS. The SQL contained in the view is passed to the DBMS, and the data is retrieved, at run time, when the view is referenced by a SAS program. A view can be treated as a "read-only" dataset for any SAS procedure. The following PROC SQL statement creates the view MYLIB.NCACCTS, which is later referenced in the PROC PRINT statement.

```sas
PROC SQL;
CONNECT TO ODBC (REQUIRED="DSN=SALES");
CREATE VIEW MYLIB.NCACCTS AS
SELECT * FROM ACCOUNTS WHERE STATE = "NC"
```

```sas
PROC PRINT DATA=MYLIB.NCACCTS;
RUN;
```

Data may be inserted, updated or deleted using the pass-through interface. To update or delete data, an SQL statement must be constructed and executed. Since the updates and deletes are not "positioned" updates, any rows that meet the criteria for the
statement will be affected. For example, if you wanted to delete the record for the "MyCorp" account from our Oracle Sales database, you could submit:

```
PROC SQL;
CONNECT TO ODBC (REQUIRED="DSN=SALES");
EXECUTE (
    DELETE FROM ACCOUNTS
    WHERE ACCOUNTNAME = "MyCorp"
) by ODBC;
```

**CONNECTION INFORMATION AND ARGUMENTS**

The first step in accessing any ODBC data is establishing a connection to a specific data source. PROC SQL establishes connections to databases through the CONNECT statement. The information contained within the parenthesis of the CONNECT statement is used to define properties of the connection. In some cases, the information is passed directly to the ODBC driver manager. In others, the information tells the SAS/ACCESS interface which ODBC API functions to call and how to call them.

When connecting to ODBC data sources, you can explicitly specify all of the required connection information in the CONNECT statement, or information can be dynamically provided to the ODBC Driver Manager and specific ODBC drivers via dialog boxes.

The ODBC Driver Manager supports four mutually exclusive connection modes: PROMPT, NOPROMPT, COMPLETE, and REQUIRED. If a connection mode is not specified, the default of NOPROMPT is assumed.

- **PROMPT** <- <"Driver-specific-connect-string" -> displays a dialog box that lists all data sources that are configured on the current machine and enables you to select one. If a connect string is provided, its values are used as defaults. You can override any or all options, including the preconfigured options for the data source selected. When a successful connection is made, the complete connect string is returned in the SQLXML macro variable.

- **NOPROMPT** <- <"Driver-specific-connect-string" -> issues the request to connect by using only the information that is provided in the connect string. If the information in the connect string is inadequate, NOPROMPT returns an error without displaying any dialog boxes.

- **COMPLETE** <- <"Driver-specific-connect-string" -> issues the request to connect by using the connect string (if provided). If the connect string contains adequate information, the connection will be completed without any user input. However, if the connect string is incomplete or incorrect, the driver's dialog boxes will be displayed so you can enter additional information as well as change provided information in order to complete the connection. When a successful connection is made, the complete connect string is returned in the SQLXML macro variable.

- **REQUIRED** <- <"Driver-specific-connect-string" -> issues a request to connect using the connect string (if provided). If the connect string contains adequate information, the connection will be completed without any user input. However, if the connect string is incomplete or incorrect, you can enter additional information (but not change any provided information) to complete the connection.

Use the REQUIRED argument without specifying an optional connect string in order to select from all the configured data sources.

To validate a password, use REQUIRED with a connect string. In the connect string, specify a data source that requires a password. If no password, or an incorrect password, is specified in the connect string, the driver will pop-up a dialog box requesting the password.

When a successful connection is made, the complete connect string is returned in the SQLXML macro variable.

Arguments in the "Driver-specific-connect-string" 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 "Driver-specific-connect-string" may contain several options. Some options such as UID=, PWD=, and DSN= may be common across many drivers, while many options are unique to individual drivers. Driver vendors also often choose different names for the same options. Most drivers have on-line documentation that specifies what may be contained in the "Driver-specific-connect-string".

- **DSN=** <- data-source-name -> specifies the ODBC data source to which you want to connect. Data sources must be configured through the ODBC Administrator prior to use. DSN= indicates that the connection is attempted using the ODBC API function SQLConnect, which requires a data source name. DSN= optionally accepts a user ID and password (described below), but no other arguments. This option is guaranteed to be present in all drivers.

- **UID=** <- DBMS-user-id -> specifies the DBMS user ID. The UID= argument can be used only in conjunction with the DSN= argument. Not all ODBC drivers accept user IDs. This argument is optional.

- **PWD=** <- DBMS-password -> specifies the DBMS password. The PWD= argument can be used only in conjunction with the DSN= argument. Not all ODBC drivers accept passwords. This argument is optional.

**Multiple ODBC Connections**

The SQL Procedure Pass-Through facility supports multiple connections to ODBC data sources. This allows users to operate on multiple databases at the same time, and is especially useful for cross database joins. If you use multiple simultaneous connections, you must use the aliases to identify the different connections. If you do not specify an alias, the default alias, ODBC, is used. The functionality of multiple connections to the same ODBC data source may be limited by the particular data source's driver. For more information about creating connection aliases see: SAS/ACCESS Software Changes and Enhancements: SQL Procedure Pass-Through Facility, Version 6.

**ODBC Connection Examples**

These examples use ODBC to connect to a data source that is configured under the data source name Scott's Data using the alias SCOTT1. The first example uses the connect method guaranteed to be present at the lowest level of ODBC conformance. Note that DSN= names may contain quotes and spaces.

```
proc sql;
  connect to ODBC as scott1 (dsn="Scott's Data" uid=scott pwd=tiger);
```

The next example uses the connect method that represents a more advanced level of ODBC conformance. It uses the password input dialog box that is provided by the driver. The DSN= and UID= arguments are within the connect string and, therefore, are not parsed by the Pass-Through facility but instead are passed to the ODBC driver manager.

```
proc sql;
  connect to ODBC as scott1 (required = "dsn=Scott's Data;uid=scott");
```
The next ODBC example enables you to select any data source configured on your machine. The example uses the connect method that represents a more advanced level of ODBC conformance. When a successful connection is made, the connect string is returned in the SQLXMSG macro variable and can be stored if this method is used to configure a connection for later use.

```sql
proc sql;
  connect to ODBC [required];
```

This last ODBC example prompts you to specify the information required to make a connection to the DBMS. You are prompted to supply the data source name, user ID, and password in the dialog boxes that are displayed.

```sql
proc sql;
  connect to ODBC [prompt];
```

**DEBUGGING ODBC ERRORS**

In the world of interoperability, there is always the chance that products developed by different vendors do not fit together automatically on the first try. SAS Institute and other ODBC vendors have anticipated integration issues, and supply several diagnostic tools.

Probably the most robust facility available for diagnosing ODBC issues is the ODBC Spy available in the ODBC SDK (Software Development Kit) from Microsoft. The spy allows you to trace ODBC function calls made to the ODBC Driver Manager, and results returned by ODBC drivers.

The ODBC Driver Administrator, ODBC drivers and most client applications also provide tracing facilities that record pertinent messages to log files. The SAS System supports a LOG option on the ODBC CONNECT statement that causes warnings or errors returned by ODBC API calls to be written to the SAS log. This may be useful during SAS program development because some statements, such as SELECT, cause PROC SQL to make multiple calls to the SAS/ACCESS interface to ODBC, and some calls in turn must make multiple ODBC API calls. Only one error or warning may be returned to PROC SQL, so it is possible for a warning to be masked by an error or another warning. Errors cause an immediate return and PROC SQL writes them to the SAS log.

The SAS System also provides two macro variables for returning messages and return codes from ODBC drivers to SAS programs.

SQLXRC contains the return code value generated by the ODBC driver. If the contents of the string include any characters other than 'O' or spaces, an error or warning has been returned. The string contains SQLSTATE values as defined in the ODBC specification. If the first two characters of the string are '01', then the return code is a warning.

SQLXMSG contains the text of the message issued by the ODBC driver.

For example, the following code will cause the contents of the macro variables to be printed to the SAS log.

```sql
PROC SQL;
CONNECT TO ODBC(DSN="SQL Server" UID="myuserid" PWD="mypassword");
SELECT * FROM CONNECTION TO ODBC
(SELECT * FROM badtable);
%PUT &SQLXRC;
%PUT &SQLXMSG;
```

Finally, sample ODBC demo applications are available on the Microsoft ODBC SDK, that may be very helpful in verifying the proper configuration of ODBC data sources. These sample applications support simple methods for connecting to ODBC data sources and performing queries. These are excellent tools for verifying that an ODBC driver is configured properly and returning data.

**SPECIAL ODBC QUERIES**

Most relational DBMSs maintain special system tables that contain lists of the objects under the control of the DBMS. Generally, users are able to execute queries against these tables to obtain lists of available tables, columns, procedures, and other useful information. The format and names of these system tables varies from DBMS to DBMS, so it is not possible to write generic ODBC SQL queries to read system information from all possible data sources. In fact, some ODBC data sources such as spread sheets and text files will not have system tables at all. To provide ODBC client applications with the ability to get this information, the ODBC API supplies a number of special functions that, when executed, will return system information in the form of an SQL query result set. The result set for one of these functions always takes on the same form regardless of the data source.

The SAS/ACCESS interface to ODBC provides access to these functions through a series of special queries. The general format of the special queries is:

```
ODBC::SQLAPI ""parameter 1", "parameter 2", "parameter 4"
```

where

- ODBC:: is required to distinguish special queries from regular queries.
- SQLAPI is the specific ODBC API function being called.
- "," parameter n" is a quoted string delimited by commas.

Within the quoted parameter string, two characters are universally recognized: the percent sign (%) and the underscore (_). The % matches any sequence of zero or more characters; the underscore represents any single character. Each driver also has an escape character that can be used to place characters within the string. Consult the driver's documentation to determine the valid escape character.

The values for the special query arguments are DBMS-specific. For example, you supply the fully-qualified table name for the "Qualifier" argument. In dBase, the value of "Qualifier" might be "c:\\dbase\tst.dbz" and in SQL Server, the value might be "test.customers". In addition, depending on the DBMS you are using, valid 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 may not be available for all ODBC drivers and some require a higher level of ODBC API conformance.

The following special queries are supported:

```
ODBC::SQLTables < "Qualifier", "Owner", "Table-name", "Type" >
```

returns a list of all the tables that match the specified arguments. If no arguments are specified, all accessible table names and information are returned.
ODBC::SQLColumns < "Qualifier", "Owner", "Table-name", "Col-name" >
returns a list of all the columns that match the specified arguments. If no arguments are specified, all accessible column names and information are returned.

ODBC::SQLColumnPrivileges < "Qualifier", "Owner", "Table-name", "Col-name" >
returns a list of all the column privileges that match the specified arguments. If no arguments are specified, all accessible column names and privilege information are returned.

ODBC::SQLForeignKeys < "FK-qualifier", "FKowner", "FK-table-name", "FK-qualifier", "FKOwner", "FKTableName" >
returns a list of all the columns that comprise foreign keys that match the specified arguments. If no arguments are specified, all accessible foreign key columns and information are returned.

ODBC::SQLPrimaryKeys < "Qualifier", "Owner", "Table-name" >
returns a list of all the columns that comprise the primary key that matches the specified table. A primary key can be comprised of one or more columns. If no table name is specified, this special query will fail.

ODBC::SQLProcedureColumns < "Qualifier", "Owner", "Proc-name", "Col-name" >
returns a list of all the procedure columns that match the specified arguments. If no arguments are specified, all accessible procedure columns are returned.

ODBC::SQLProcedures < "Qualifier", "Owner", "Proc-name" >
returns a list of all the procedures that match the specified arguments. If no arguments are specified, all accessible procedures are returned.

ODBC::SQLStatistics < "Qualifier", "Owner", "Table-name" >
returns a list of the statistics for the specified table name, with options of SQL_INDEX_ALL and SQL_ENSURE set in the SQLStatistics API call. If the table name argument is not specified, this special query will fail.

ODBC::SQLTablePrivileges < "Qualifier", "Owner", "Table-name" >
returns a list of all the tables and associated privileges that match the specified arguments. If no arguments are specified, all accessible table names and associated privileges are returned.

ODBC::SQLGetTypeInfo
returns information about all data types supported by the data source.

For additional information concerning connection options or special queries, see: SAS/ACCESS Software Changes and Enhancements: SQL Procedure Pass-Through Facility, Version 6.

CONFIGURING ODBC DATA SOURCES

To actually use ODBC to access data, a data source must be defined to the ODBC driver manager. This process can take place during the installation of most ODBC drivers or after the driver has been installed. The result of the configuration process is an ODBC data source. The configured data source is identified by a Data Source Name (DSN) which you specify during the configuration. For example, a DSN named "Yearly Sales" could be created specifying that the data resides in an SQL Server database called SALES on the server identified as NTServer. Additional information such as a userid and other options can also be associated with a DSN. The DSN is used by the client application (such as the SAS/ACCESS Interface to ODBC) to identify the ODBC data source to which it would like to connect. The driver and driver manager then use the information that was configured with the DSN to establish that connection.

When configuring an ODBC data source you should keep in mind that the ODBC driver must surface the data source in a relational DBMS model. In other words, the driver must appear to connect to a database and queries are then run against tables in that database. Furthermore, tables are rectangular consisting of rows and columns where a column has only one associated data type.

Obviously, relational DBMS data sources have no problems in fitting this model. However, some non-relational DBMS data sources and single tier files (spread sheets) do not fit this model. ODBC drivers for these data sources must force some conventions or restrictions on the use of these data sources. These restrictions vary from data source to data source and driver to driver, so consult the documentation that is provided for the ODBC driver.

In addition to the data model, you should also consider what type of driver is being used for the data source. If your data source is simply a file (such as a DBASE .DBF file) that is not being managed by DBMS software then you have a single tiered ODBC data source. Configuration of the data source then consists primarily of identifying the directory or file where the data exists. If you are a multi-tiered data source, you must ensure that the DBMS software is up and running and the DBMS client software is installed on your machine in addition to configuring the ODBC data source.

CONFIGURING AN EXCEL 5 DATA SOURCE

Drivers for Microsoft's Excel 5 (and above) can generally be considered single tiered ODBC drivers. As such, they should be simple to configure because there is no DBMS software with which to contend. However, like most spread sheets, its data does not conform to the relational DBMS model used by ODBC. For example, though the spread sheet form appears rectangular, its data is not; columns are not restricted to a single data type; and there are multiple spread sheets in an .XLS file.

To map Excel 5 data to ODBC, the Microsoft ODBC driver for Excel 5 has made the following conventions:
- An .XLS file is treated as an ODBC database.
- Spreadsheets and named ranges are treated as ODBC tables.
- Column data types are determined by scanning the rows of data in the named range or spread sheet and determining the type that is used most often. The amount of scanning is configurable while configuring the data source.
- Column names are determined from the first row of data in the named range or spread sheet.

To create an ODBC data source, you must use the ODBC administrator program. This program is located in the control panel on Windows platform. (For non-Windows platforms, the ODBC administrator program may not exist, or may be in a different group. Consult the documentation that accompanied the driver.) For Windows platforms, the ODBC administrator program is generally titled "ODBC". However, on 32-bit platforms such as Windows NT and Windows 95 you may have the choice of 16 or 32 bit ODBC drivers. To setup 32-bit ODBC drivers, the ODBC Administrator may be titled "32bit ODBC" or will have a "32" somewhere in the program icon. This example will use the 32-bit Administrator and driver.
Database Management Facilities

Figure 5: NCSALES.XLS Workbook

For the following example we will be creating a data source for an Excel 5 .XLS file called NCSALES.XLS. This workbook contains two spreadsheets "RepSales" and "CountyRev". The workbook is represented in figure 5 above. To configure an ODBC data source for this workbook, perform the following:

1. Start the ODBC administrator program and you will be presented with the "Data Sources" window. This window shown in figure 6 lists all the Data Source Names currently configured. From this list you can update a data source or create a new one.

   The "Options..." button is used for tracing ODBC calls. These traces can prove helpful in analyzing performance or driver problems.

   The "System DSN..." is used to create DSNs that are local to a particular machine, but may be used by multiple users on that machine. In contrast, non-system DSNs are only valid for the user that created them.

   To add a new data source, simply choose the "Add..." button.

2. Figure 7 shows the next window that is displayed, the "Add Data Source window". This window lists all of the currently installed ODBC drivers. In our example we will select Microsoft Excel Driver (*.xls), and choose "OK".

3. The Microsoft Excel driver setup dialog is displayed next. In this box you specify the Data Source Name by which the data source will be identified in client applications and a description. To identify the ODBC database, you should indicate what version of Excel file is being used, and (for Excel 5) specify the workbook containing the data.

4. By completing the Excel Setup window as shown in step 3, you could close the window and then have a complete data source. Or you can specify some additional options. These options are driver specific and pertain only to the Excel driver; other drivers will have other options.

   For Excel 5 there are two options, "Rows to Scan" and "Read Only". The "Rows to Scan" option tells the driver how many rows of Excel data to examine to determine the data type for a specific ODBC column. Remember that the ODBC driver must project the Excel data into the relational DBMS model. Excel does not restrict the data in a column to only one type so, the ODBC driver must look at the data in the column and decide what type of data is most common. If, during a query, the data found in a column does not have the same type as that determined by the scan, that value will be returned as NULL.
Database Management Facilities

5. After closing the Excel Setup window, the Data Sources window is redisplayed, this time listing the new "NC Sales" data source just created.

The data source setup is now complete and can be accessed by any ODBC client application including SAS, ACCESS. To access the data in the "RepSales" sheet, you would submit the following SAS statements:

```
PROC SQL;
    CONNECT TO ODBC (DSN="NC Sales");  
    SELECT * FROM RepSales;
```

In the Microsoft Excel 5 driver, spreadsheets within the .XLS file are considered system tables and named ranges are treated as tables. To reference an entire spreadsheet in an ODBC SQL statement you must add a 's' (dollar sign) to the end of the name and double quote the resulting name. If you are referencing a named range, this is not required. If you are using non-Microsoft drivers you may not be able to use a spreadsheet as a table. You will need to create a named range (list) in the spreadsheet and refer to that as the table name.

**EXAMPLE OF AN SQL SERVER DATA SOURCE CONFIGURATION**

In comparison to the Excel 5 data source configuration, consider the following dialog box for the Microsoft SQL Server driver setup. The SQL Server driver is a multi-tiered driver. The driver itself does not manipulate the disked data, but must communicate with the SQL Server. This server may be on the same machine as the driver, but most likely on another machine. Because of the networked nature of this data source there are a number of things that must take place prior to configuring the driver.

- SQL Server - Server must be installed and running.
- Networking connectivity between the Server and Client machine must be established.
- SQL Server - Client software must be installed and configured on the client machine.
- User must be identified and given permissions to use data under control of the server.

Once the tasks above have been completed, the task of configuring the driver for SQL Server is similar to that of Excel. The differences come in the Driver Setup window. These windows collect information specific to a driver for defining a data source. Figure 11 below shows an example configuration used for accessing a Microsoft SQL Server 6.0 server on a Windows NT workstation.

```
[Diagram of SQL Server ODBC Driver Setup]
```

In the configuration above, most of the fields were left at their default values. These values are determined from the configuration of the SQL Server client software that is installed separately from the driver. This will not always be the case. For instance, you may want to access a server that has not been defined in your local SQL Server client configuration. In this case, you would specify the information necessary to connect to the server including the networking address and network library to be used for the network protocol.

While the Microsoft SQL Server ODBC driver is a multi-tiered driver, it is different from many other drivers on the market. Most multi-tiered drivers must call DBMS provided APIs to service their data needs. In the case of SQL Server drivers, most vendors must make calls to the SQL Server DB-Library API. However, Microsoft’s SQL Server driver does not call the DB-Library API, it is a native DBMS client itself. It uses many of the same DLLs that support the DB-Library API. This has allowed Microsoft’s driver to eliminate an extra level of calls in its ODBC driver and has allowed them to write a driver whose internals are optimized for ODBC functionality and performance while not being limited by the DBMS client API. This type of native driver allows ODBC client applications to attain performance equal to or better than that achieved by applications written to use the DB-Library API.

**CONNECTING TO AN ODBC DATA SOURCE WITHOUT CONFIGURING A DSN**

Sometimes it is inconvenient or undesirable to create a DSN for every data source that you need to access. There are a number of
methods that may be used to dynamically connect to a data source without first creating a DSN with the ODBC administrator. All of these methods involve the parameters passed to the CONNECT TO ODBC statement in PROC SQL.

To interactively connect to a data source, you can specify the PROMPT or COMPLETE options on the CONNECT TO ODBC statement. Both of these options can be used to dynamically select a preexisting DSN and then override any of the default values configured for it. For example, you could use the prompt option to select the NC SALES DSN that we configured for Excel 5 and then choose a different "XLS file to be used during the connection. This changes set not saved in the DSN, but is active only for the life of the connection. Any option that can be configured for a DSN may be overridden in this way.

To access an undefined database non-interactively, you can use an existing DSN and override the values associated with it. For example, assume you have a Sybase System 10 DSN called "Research" which is used to access the "XYZ" database on the "NOReach" server. If you are using the interos driver and want to access the "Marketing" database on the "NCOMKT" server you could connect to the database using:

```sql
PROC SQL;
CONNECT TO ODBC("DSN=Research;Server=NCOMKT;DB=Marketing"
                 PWd=mpassword");
```

The userid already associated with the Research DSN will be used to access the new server and database.

Consider the different driver setups that we examined in the previous two sections. Drivers generally provide keyword options that may be specified in a connect string for many of the values needed to define a DSN. As you can imagine, there are many connection string options. These options vary from driver to driver and vendor to vendor. These options are often documented with the string driver. For example, the following options are used by the Microsoft Excel and SQL Server drivers that were used in previous examples:

- **DSN=** is used to specify the database qualifier for Microsoft's ISAM drivers such as Excel 4 and 5 and ACCESS. For an Excel 5 driver this would be an .XLS file.
- **UID=** is used to specify the data source name in a connection.
- **PWD=** specifies the userid to be used for the connection.
- **DRIVER=** specifies the password to be used for the connection.
- **SERVER=** specifies the name of the database server to which you want to connect.
- **DATABASE=** specifies the SQL Server database that will be used in the connection.

### COMMITMENT CONTROL

ODBC drivers support two modes of transaction control: auto-commit and manual-commit. By default, if a driver supports the ODBC extended function SQLSetConnectOption() then the default commit mode is auto-commit. This is typically the case and therefore the default when using the SAS/ACCESS interface to ODBC.

In auto-commit mode, each SQL statement is committed when it finishes executing. In manual-commit mode the client application must call the SQLTransact() ODBC API function. To allow you to control ODBC transactions, the following special features have been added to the SAS/ACCESS interface to ODBC in Release 6.11 of the SAS System.

**NOAUTOCOMMIT** is an option that can be specified in the PROC SQL CONNECT TO ODBC statement. It will cause the transaction mode for the connection to be manual-commit.

```sql
ODBC::SQLTransact "ROLLBACK" | "COMMIT"
```

This statement is used with the PROC SQL EXECUTE statement. When the connection that is executing the statement is in manual-commit mode, this statement will commit or rollback any previously uncommitted statements performed by the connection.

```sql
ODBC::SQLSetConnectOption "autocommit", "off" | "on"
```

This statement is used with the PROC SQL EXECUTE statement and controls the commit mode for an existing connection. To switch a connection to manual-commit mode, submit the following:

```sql
EXECUTE (
    ODBC::SQLSetConnectOption "autocommit", "off" ) by ODBC;
```

To switch back to auto-commit mode, just execute the code again using "on" rather than "off" as the parameter.

The following is an example of how these statements might be used. Please note the use of the SAS macro facility to support conditional execution of SQL Pass-Through statements.

```sql
$macro transact;
$if (&sqlxrc = 0) $then {
$put "Changes Committed";
execute (ODBC::SQLTransact "COMMIT") by ODBC;
$end;
$else {
$put "Changes Rolled Back";
execute (ODBC::SQLTransact "ROLLBACK") by ODBC;
$end;
$end;
```

```sql
PROC SQL;
/* connect with manual-commit mode */
CONNECT TO ODBC (NOAUTOCOMMIT DSN="SALES" PWd="TIGER")
/* delete some rows */
EXECUTE (DELETE FROM ACCOUNTS
WHERE ACCOUNTNAME = "MyCorp"
) BY ODBC;
/* use the transact macro to commit or */
/* rollback the transaction. */
$transact;
/* put connection back into auto-commit mode */
EXECUTE (ODBC::SQLSetConnectOption "autocommit", "on") BY ODBC;
```

In the lines above, we established a connection to the Oracle "SALES" data source, specifying the Oracle password so that the connection is made without prompting by the driver. The NOAUTOCOMMIT connection option is used so that the connection is established in manual-commit mode. Following the connection, an attempt is made to delete records from the ACCOUNTS table. The %transact macro is used to check the SQLXRRC variable to see if the Pass-Through statement executed without error or warnings. If there was no error the changes will be committed, otherwise they will be rolled back. Finally, the connection to the Oracle data source is placed back into auto-commit mode by executing the ODBC::SQLSetConnectOption turning autocommitt on.
CONCLUSION

This paper has reviewed many technical concepts associated with the ODBC specification and its use by the SAS/ACCESS interface. We started with an overview of the components of ODBC and their inter-relationships. The difference in driver types was explained as was the expectations of different levels of driver conformance. We then looked at how to use PROC SQL Pass-Through statements with the SAS/ACCESS interface to ODBC. ODBC connection options were detailed along with the special ODBC:: queries that may be used to retrieve data source specific system information. After that, we configured data sources for Excel 5 and Microsoft SQL Server as examples of single-tier and multi-tier driver configurations that can be used with the SAS System. We discussed how to use special driver connect string options to connect to a data source without first creating a DSN. Finally, transaction control was detailed with examples of how to control commits in your SAS System ODBC applications. With the information presented in this paper you should have a very complete understanding of the workings of the SAS/ACCESS interface to ODBC and how to manipulate the interface to meet your needs.

TRADEMARKS

SAS and SAS/ACCESS are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. AS/400, DB2, and OS/2 are registered trademarks or trademarks of International Business Machines Corporation. ORACLE is a registered trademark or trademark of Oracle Corporation. © indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

REFERENCES


ACKNOWLEDGMENTS

Thanks to Greg Shanker for his input to this paper, including his "editorial" review and presenting this paper at SUGI 21.

AUTHOR

Forrest Boozer
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
PH: (919) 677-8000
e-mail: SASJFB@UNX.SAS.COM