

Paper 135-2008

Getting SAS[®] to Play Nice With Others: Connecting SAS[®] to Microsoft SQL Server Using an ODBC Connection

Candice Riley, RAND Corporation, Santa Monica, CA

ABSTRACT

Getting SAS to interact with other databases is often vital to the success of a project, but this can be a problem when SAS and the databases being used are on different platforms. This paper discusses connecting SAS to a Microsoft SQL Server from both PC and Linux environments using SAS/ACCESS and the odbc engine. The paper also discusses the role the Libname Statement and the Pass Through Facility play in this connection. Proc SQL, macro variables, and some basic SQL settings and syntax will be discussed to demonstrate the power of this connection. Finally, UnixODBC and FreeTDS are discussed as third-party tools in getting the Linux environment set up.

KEYWORDS

- SAS/ACCESS
- PROC SQL
- Microsoft SQL Server
- ODBC

INTRODUCTION

SAS is a very powerful analytical tool, but not all data comes as a SAS dataset. Being able to get data into and out of SAS is an integral part of the work we do with clients at the RAND Corporation. Additionally, in our group at RAND, we keep our data warehousing environment and our analytical environment on separate platforms to maximize resources for both, which has made communication between SAS and MS SQL Server challenging. One of the workarounds we used was to export and import all the data through intermediate csv files, but doing so had many obvious drawbacks, including efficiency and data integrity. The ideal situation would be one where we could access all the SQL data, the SQL relationships, and the SAS analytical power from one program in one language. With the help of SAS/ACCESS, an odbc connection, and a couple of third-party tools, we were able to do all of this and more. Our machines are now seamlessly sharing data across platforms and we never have to leave the SAS environment. Here, we discuss what we have done, first in the PC environment and then in the Linux environment.

THE PC ENVIRONMENT

Our project members have their own Windows desktop or laptop running PC SAS. Giving each of them the power to connect to the SQL Server through their SAS programs would be very liberating. Instead of relying on the csv files provided, they could create their own data cuts in real time and never worry about how recently the intermediate files were created. So, the first connection established was between the desktop PC and the MS SQL Server 2005 running on a Windows 2003 Server. This was a natural starting point for testing SAS/ACCESS because the PCs have a standard odbc wizard, which makes connecting straightforward.

SETTING UP THE ODBC & DSN

For the windows PC machines, Microsoft has installed an ODBC Data Source Administrator under the Administrative Tools that come standard. This is where you create the odbc connection and assign a Database System Name (DSN) to it, which is how you tell SAS which odbc connection to use. There are both User DSNs and System DSNs, depending on what sort of access should be granted to this connection. A User DSN will only give that user access, while a System DSN will give any user on the computer access to use that DSN. Once you decide which type to set up, you click ADD and follow the wizard to set up the connection. For this paper, we use the DSN "MyDSN", and it will be connecting to MyServer.

USERNAMES AND PASSWORDS

During set up of the SQL Server, you can choose between different types of authentication. If you choose Windows Authentication, this means users will use their Network logins to get access to the SQL server. In other words, users can write their SAS code without typing a Username or a Password, and the Windows servers will authenticate as long as they are running the code from the machine they are logged in to. Another choice is SQL Authentication. This is where usernames and passwords are created independently of any network or windows logins. In this case, it will be important to make sure users adequately protect their username/password information, because they will have to put that information in SAS code to connect. For this paper, my SQL has been set up with Mixed Mode

Authentication, meaning that both Windows and SQL Authentication are activated and that both types of logins will be accepted. For our PC SAS examples, we will show Windows Authentication.

DATA CONNECTION: LIBNAME

Now that the settings have been established, we can begin connecting to the SQL Server. Because we assume Windows Authentication for these PC examples, the username and password do not need to be given. SAS/ACCESS allows two possibilities for reading and writing data with the SQL Server. One of these possibilities is through the Libname statement. In this case, the connection string is supplied during the libname statement:

```
LIBNAME mylibref ODBC
NOPROMPT="UID=;PWD=;DSN=MyDSN;SERVER=MyServer;DATABASE=MyStore;" ;
```

The first part of the statement proceeds as usual. We declare "libname", then give it a libref, which we will use from here on out to access our data through the two-level naming structure. The next piece, "odbc," lets SAS know it will be using the odbc engine. The "noprompt" option passes all of our connection information in the string at once for batch processing. As mentioned, our uid and pwd are empty because we are using Windows Authentication. We are using the DSN "MyDSN", which we created above to connect to MyServer. And on MyServer we want to connect to the MyStore database.

The next piece of code shows how we reference the data:

```
PROC PRINT DATA = mylibref.grains(obs=10);

DATA expensive_fruit;
SET mylibref.fruit;
If price>5;
```

We can now use the SQL dataset as if it were any other library referenced SAS dataset. Above, we have two examples where we read data from the MyStore database on MyServer. We do a PROC PRINT to see the first ten lines of data of our Grains table. We also create a work dataset called Expensive_Fruit, which is a subset of our SQL Fruit table. We can now use these datasets to run analysis or write reports or tap into any of the functionality SAS provides. This is incredibly powerful. We can read all our SQL tables without ever leaving the SAS environment using very familiar SAS code.

Below, we will see that we can write to the SQL Server as well.

```
DATA mylibref.veggies;
SET farmers_market;
if type='veggie';

PROC SQL;
INSERT INTO mylibref.fruit(fruit, farmer, price)
SELECT fruit, farmer, price
FROM farmers_market_supply
WHERE type='fruit';
QUIT;
```

In these examples, we are writing data to our MyStore Database from local work SAS datasets. The first example shows how you can create a new table called Veggies in MyStore using familiar Base SAS syntax. The second shows how you can add data from a SAS work file to the pre-existing Fruit table, using PROC SQL. We can now move data between these two platforms seamlessly.

This is incredibly convenient, because we can have our data in its native format but still tap into the power of SAS, and we can do this without importing or exporting our data through temporary intermediate files. We can read data from the SQL Server and write data back to it without leaving the SAS environment. And the libname capability allows you to keep using the SAS language and SAS/ACCESS does the translating into SQL code for you.

DATA CONNECTION: PASS THROUGH FACILITY

The other option SAS/ACCESS offers is called the Pass Through Facility. The Pass Through Facility uses Proc SQL and requires all the code that will be passed to the SQL Server to be written in the SQL language (in this case, Transact-SQL). This allows the user to tap into the relationships, indexes, and other settings supplied by MS SQL

and have the resulting dataset returned in SAS format. In this case, the connection string remains similar to the Libname version.

```
PROC SQL;
CONNECT TO ODBC as myODBC (NOPROMPT= "DSN=MyDSN;SERVER=MyServer;UID=;PWD=;
DATABASE=MyStore");
CREATE TABLE calif_fruit AS
SELECT * FROM CONNECTION TO myODBC
(SELECT f.*
FROM fruit as f inner join vCA as c
on f.fruit=c.item
ORDER BY f.fruit, f.price, f.qty
);
QUIT;
```

In the above code, we start with setting up a Proc SQL statement. The first line declares that we will be Connecting to another Server using the odbc engine. Again, the noprompt option allows us to supply the same connection string with all the relevant information at once for batch processing. The next couple of lines are the Proc SQL statements for creating a new SAS table, this one is a work dataset called calif_fruit, and it will be populated with all fields that get returned from our connection. Next, between the parenthesis, we supply the SQL code that is going to be passed to the MyStore database. Here, we perform an inner join between the fruit table and the view vCA, keeping only fruit fields from rows of data that match the inner join. The resulting dataset is only fruit that is grown in California and can now be used as any other SAS work dataset.

As we just saw the Pass Through Facility offers similar read/write functionality as the Libname statement. But because we are able to successfully pass raw SQL code, you would expect to be able to tap into some of the other SQL features. In the code below, we demonstrate this power.

```
PROC SQL;
CONNECT TO ODBC as myODBC (NOPROMPT= "DSN=MyDSN;SERVER=MyServer;UID=;PWD=;
DATABASE=MyStore");
EXECUTE (
sp_CalcRevenue)
BY myODBC;
QUIT;
```

Here, we use a PROC SQL EXECUTE statement to pass non-query SQL statements. Execute statements are passed exactly as written and are useful for statements such as UPDATE, INSERT, DELETE and also for executing stored procedures. No SAS dataset is created with the Execute statement; therefore, we do not need the "create table" syntax surrounding it. The action is performed on the SQL Server and then the SAS program proceeds to the next series of statements. In this case, we execute a stored procedure called sp_CalcRevenue. It will run through all its stored code and any changes it has made or tables it has created will be accessible to our SAS program after it has completed running.

This demonstrates the wide range of SQL functionality that SAS users can now access. Stored Procedures can be used for back-end processing, to maintain data integrity, and for other housekeeping jobs. Being able to use the Execute statement and run more than just data queries really gives SAS users a powerful interface for managing SQL data.

THE LINUX/UNIX ENVIRONMENT

The next connection we established was getting the Linux server and the SQL Server to communicate. While project members have PC SAS running on their laptops for quick ad-hoc reports, they also have SAS 9 on the Linux machine for running models, simulations, and other large processes that churn years and gigabytes of data. Getting our Linux server connected to our SQL Server would allow these batch processes to take advantage of the speed and power of the Linux machine, while also only passing data over the network when required by the code. Also, by having these important programs on the Linux machine, this puts them in a central location where each member of the team can access and alter them as needed. But unlike with PC machines, the Linux system does not come standard with an odbc manager wizard. Thus, to properly set up our odbc connection, we need to look at a couple of third-party tools.

SETTING UP THE ODBC & DSN

To set up the Linux server, both an odbc driver manager and an odbc driver for MS SQL Server need to be installed. In this paper, we use unixODBC as our driver manager. We also need an odbc driver, which will give us the DSN name for the SQL Server connection. For this paper, we chose FreeTDS (v8.0) as the SQL Server driver, but there are others, such as EasySoft, that can also be used. Once both of these utilities have been installed on the Linux machine, you can then begin setting up the connection to your SQL Server.

Two main files need to be created in our home directory. The first is the tds.driver.template, which will tell unixODBC about the FreeTDS driver we intend to use. This will point to the file libtdsodbc.so that was created when we installed FreeTDS. The template looks like this:

```
[FreeTDS]
Description = v0.63 with protocol v8.0
Driver      = /usr/local/lib/libtdsodbc.so
```

The second is the tds.datasources.template, which we use to set up the DSN. We use this template to enter the driver entry into the .odbc.ini file. This template looks like this:

```
[ODBC Data Sources]
MyDSN = SQL Server

[MyDSN]
Driver      = /usr/local/lib/libtdsodbc.so
Description = SQL Server
Trace       = No
Server      = MyServer.rand.org
Database    = MyStore
Port        = 1433
TDS_Version = 8.0

[Default]
Driver      = /usr/local/lib/libtdsodbc.so
```

Again, our DSN will be used to inform SAS which odbc connection we want to use. In the file above, we point to the FreeTDS driver we installed (the same one in our driver template), to the server we are referencing either through name or ip address, and to the database on the server we will be using. We specify port 1433 for SQL Servers and TDS version 8.0. This will create a DSN called MyDSN for the MyStore database on MyServer. UnixODBC gives us the tool "odbcinst" to set up these configurations pretty quickly and easily.

USERNAMES AND PASSWORDS

We cannot use Windows Authentication for our SQL Server from a Linux server, because the users logged into Linux are not logged in to their Windows account. So, for these examples, we will be using the SQL Authentication of our Mixed Mode set up. We do this by creating a user on the SQL server side. In the following examples, our user, "linux_user," will have tightly controlled permissions.

DATA CONNECTION: LIBNAME

Now that the set up has been taken care of, we should be able to access our data through the Libname statement in the same way PC SAS was able to.

```
LIBNAME mylibref ODBC DSN=MyDSN USER=linux_user PASSWORD=sqlpwd SCHEMA=sales;

PROC CONTENTS DATA=mylibref._all_;

DATA mylibref.sales012008;
SET farmers_market_sales;
IF month(date)=1 and year(date)=2008;
```

The libname statement is set up similarly to its PC counterpart. The difference here is that the Server and the Database have been specified in our DSN declaration, so we do not need to repeat it. But since we can no longer rely on Windows Authentication, we must now pass a user and password field. Also, we have added the Libname Option SCHEMA. The default for this option is none, although when a user is created on the SQL side, the default schema is usually set to "dbo". In this case, we wanted to access the schema "sales" specifically. This is really

important, because it demonstrates how the intricacies of our SQL environment are not lost, even though we are accessing that environment from the outside. Once the libname is set up, we can reference the SQL Server tables just as easily as if they were any two-level SAS dataset.

After the libname statement above, we have a PROC CONTENTS grabbing data for all the tables in the sales schema. This will not grab information from the dbo schema; instead, it will return a focused report on the just the sales tables. Again, we see that we can successfully read across our connection.

The next few lines of code create a new table in the MyStore database in the sales schema that should consist of all of the farmers market sales for January 2008. This example works similarly to its PC counterpart, where the two-level naming convention works seamlessly to write data to our SQL Server. We can now read and write from the SQL Server to the Linux box just as easily as we could from the PC to the SQL Server.

DATA CONNECTION: PASS THROUGH FACILITY

The final piece is the Pass Through Facility on the Linux server. Again, our connection string will be slightly different from the PC version, but it is very similar to the Linux Libname statement version.

```
%let recent = Cast('1/1/2008' as Datetime);

PROC SQL;
CONNECT TO ODBC as myODBC (DSN=MyDSN USER=linux_user PASSWORD=sqlpwd);
CREATE TABLE recent_CA as
SELECT * FROM CONNECTION TO myODBC
(SELECT f.*
FROM fruit as f inner join vCA as c
on f.fruit=c.item
WHERE date>&recent
ORDER BY f.fruit, f.price, f.qty
);
QUIT;

Proc sql;
CONNECT TO ODBC as myODBC (DSN=MyDSN user=linux_user password=sqlpwd);
EXECUTE
(UPDATE fruit
SET price = price + price * .15
WHERE fruit = 'orange')
BY myODBC;
QUIT;
```

In the first example, the inner join code from earlier is revisited and made more powerful by adding a macro variable. Here, we see that we can extend the power of SAS to non-SAS datasets. Even though we write our Pass Through Facility code in the SQL language, we can still make it dynamic and access the macro facility. But it is important to notice that our macro variable needed to be SQL code-friendly. In this case, that means that our date has to be a SQL date, not a SAS date, because the pass Through Facility does not translate the code from SAS to SQL.

In the second example, we use the PROC SQL EXECUTE statement to pass through the non-query statement UPDATE. Here, we had to update the price of oranges because of a recent frost. These two examples show that we have gotten the Pass Through Facility to both read and write data as we hoped it would.

CONCLUSION

After installing a couple of third-party tools to get our Linux environment all set up, we were able to get SAS and Microsoft SQL Server to work seamlessly across different platforms. This has provided much value added to our projects' productivity and data integrity. We can now all access the SQL Server from our personal laptops and from our communal code on the Linux box. Without ever leaving the SAS environment, we can now read and write SQL data, bringing the power of SAS to other data structures, but without giving up the relationships and intricacies provided by the SQL Server. SAS/ACCESS provides incredible power to SAS users whether they code in SAS or in the SQL language.

REFERENCES

SAS Institute Inc. (2004), *SAS/ACCESS 9.1 Interface to Relational Databases*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2004), *SAS/ACCESS 9.1 Supplement for ODBC SAS/ACCESS for Relational Databases*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2003. SAS Technical Support Document (TS-685). "Setting up SAS/ACCESS to ODBC on UNIX Platforms to Interface with ODBC-Compliant Databases." <http://support.sas.com/techsup/technote/ts685.pdf>

Harvey, Peter. unixODBC Manual. "How to use unixODBC with FreeTDS." <http://www.unixodbc.org/>

Bruns, Brian and Lowden, James K. "FreeTDS User Guide." <http://www.freetds.org/userguide/>

ACKNOWLEDGMENTS

The author would like to thank Mitch Tuller, Aimee Bower, Adrian Overton, Ken Girardini, Jan Hanley, and Scott Miller.

CONTACT INFORMATION

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

Candice Riley
RAND Corporation
1776 Main Street
Santa Monica, CA 90404
Work Phone: (310)393-0411
E-mail: criley@rand.org
Web: www.rand.org

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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