ACCESSING YOUR DATABASES WITH THE SAS® SYSTEM

C. Michael Whitney. USAF Environmental Technical Application Center

ABSTRACT
This paper will discuss SAS® 6.07's improvements to ACCESS and SQL and how they eliminate the need to manually create access and view descriptors in order to retrieve from and update DB2® databases. Techniques described can be applied to other relational databases. Several of SAS SQL's other features will also be covered, including SQL and macro variables, summary functions, the case expression, and multi-table selects.

INTRODUCTION
The U.S. Air Force Environmental Technical Application Center (ETAC) has over 6000 DB2 tables that are accessed on a daily basis. Of all the features that SAS provides, the most used at ETAC are its database access procedures, PROC ACCESS and PROC SQL. In earlier versions of SAS, we were forced to manually create a new access and view descriptor for each table we wanted to use with SAS, and store these in a library. When the databases underwent a change, it necessitated a change of several hundred access and view descriptors. With the release of SAS 6.07, we were freed from this slow drudgery. The tools that liberated us were ACCESS's new ability to create access and view descriptors on the fly within our programs, and SQL's. new Pass-Through Facility. SQL also has the added bonus of being able to manipulate a lot of the data in a wide variety of ways without extra SAS coding.

NEW ACCESS TECHNIQUES
Release 6.07 of SAS was the first to allow the creation of access and view descriptors from within the code, instead of interactively. This allows for the creation of temporary descriptors that can be created in the work library, and gotten rid of when SAS is finished, instead of keeping a huge permanent library of descriptors.

PROC ACCESS OBMS=OB2;
CREATE WORK.SCOTTWNO.ACCESS;
SSID = DSN;
TABLE = AOB.SFC724338;
ASSIGN = YES;
CREATE WORK.SCOTTWNO.vIEW;
SELECT YEAR MO DAY HR WSPD WOIR;
SUBSET WHERE YEAR BETWEEN 1971 AND 1991;
RUN;

Data gathered from the SCOTTWNO view looks like this:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MO</th>
<th>DAY</th>
<th>HR</th>
<th>WSPD</th>
<th>WDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>125</td>
</tr>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>120</td>
</tr>
</tbody>
</table>

In the first code portion, the access descriptor is created. Following is its breakdown:

- DBMS=OB2: - Tells Access which database to use. In this case, DB2.
- SSID = DSN: - This is needed when interfacing DB2 and specifies the subsystem ID. It can be omitted if your system has the DB2SSID= option set.
- TABLE = AOB.SFC724338: - Another necessary line for OB2. This specifies the DB2 table or view that will be used to create the SAS access descriptor. The authorization ID (AOB in this case) is required in batch mode, but it's a good idea to always use it.
- ASSIGN = YES: - Creates SAS variables based on the first eight characters of the database column names when creating an access descriptor. A YES value automatically resolves any duplicate values, but you can only modify them during the access descriptor creation. It also means that any view descriptors created using this access descriptor will automatically use the new SAS variable names. A NO is the default, and allows modification in both access and view creation.

The next code portion creates a view descriptor based on the newly created access descriptor SCOTTWNO.ACCESS.

CREATE WORK.SCOTTWNO.VIEW; - Creates a view based on the previously created access descriptor.
SELECT YEAR MO DAY HR WSPD WDIR; - Selects the columns we want from the database. An ALL can be used as well, to include all of the columns from the database.
SUBSET WHERE YEAR BETWEEN 1971 AND 1991; - Subsets the data, based on the database column names, by using the WHERE, ORDER BY, HAVING or GROUP BY commands.

RENAMEING AND FORMATTING VARIABLES
While the ASSIGN command does a good job of converting the database column names to eight character SAS variable names, it does so by truncation. This can lead to some confusion, and it's a better idea to rename them to something more meaningful. This is accomplished with the RENAME command. A FORMAT statement can also be used to assign a SAS variable to a particular column name.

The following code shows the creation of access and view descriptors from a DB2 database that contains weather station info with long column names. We only want stations located in Illinois, and so will use a SUBSET command to place a limit on the data returned by the view.

YEAR | MO | DAY | HR | WSPD | WDIR |
1971  | 1  | 1   | 3  | 4    | 125  |
1971  | 1  | 1   | 2  | 8    | 120  |
PROC ACCESS DBMS=DB2;
CREATE WORK.STNINFO.ACCESS:
SSID = DSN;
TABLE = ADB.SFC_INFO;
ASSIGN = YES;
RENAME STATION_NAME = BLKSTN
STATION_ELEV = STNELEV
PRECIP_RATE = PRCPRATE;
FORMAT PRECIPI_RATE = PERCENT10;
CREATE WORK.STNINFO.VIEW:
SELECT STATION_NAME LAT LON
STATION_ELEV STATE;
SUBSET WHERE STATE = 'IL';
RUN;
Data gathered from the STNINFO view would look like this:
<table>
<thead>
<tr>
<th>BLKSTN</th>
<th>STATE</th>
<th>LAT</th>
<th>LON</th>
<th>STNELEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>40300</td>
<td>IL</td>
<td>6408</td>
<td>2154</td>
<td>61</td>
</tr>
<tr>
<td>40180</td>
<td>IL</td>
<td>6356</td>
<td>2236</td>
<td>54</td>
</tr>
<tr>
<td>724338</td>
<td>IL</td>
<td>3833</td>
<td>8951</td>
<td>138</td>
</tr>
</tbody>
</table>
RENAME STATION_NAME = BLKSTN... - renames the column to the specified SAS variable name.
FORMAT PRECIP_RATE = PERCENT10... - uses a SAS format to modify a database column as it is being read.
SELECT STATION_NAME LAT LON... - selects columns in view from access descriptor. Note that the column names are used here; variables generated will use the names specified in the RENAME statement in the access descriptor.
SUBSET WHERE STATE = 'IL'; - DB2 SQL WHERE clause syntax is used to limit the data returned by specifying the column name (STATE in this case).

LIST STATEMENT

If you forget the column names of a database, you can use the LIST ALL command in either the access or view descriptor creation code. Here we display the column names found in the Scott AFB surface data table:

PROC ACCESS DBMS=DB2;
CREATE WORK.SCOTTSFC.ACCESS;
SSID = DSN;
TABLE = ADB.SFC724338;
LIST ALL;
RUN;

The following data is output to the log:

<table>
<thead>
<tr>
<th>Item</th>
<th>Column Name</th>
<th>SAS Name</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JULHR</td>
<td>JULHR</td>
<td>11.0</td>
</tr>
<tr>
<td>2</td>
<td>YEAR</td>
<td>YEAR</td>
<td>6.0</td>
</tr>
<tr>
<td>3</td>
<td>MO</td>
<td>MO</td>
<td>6.0</td>
</tr>
<tr>
<td>4</td>
<td>DAY</td>
<td>DAY</td>
<td>6.0</td>
</tr>
</tbody>
</table>

Another form of this command, LIST VIEW, can be used in the view descriptor code to list all the database columns from the access descriptor that will be used in the view. Its output is identical to the LIST ALL, with the exception that only those columns used in the view are displayed, and * SELECTED * is shown at the end of the output information.

ACCESS AND MACRO VARIABLES

Access code can be made more flexible with the use of macro variables, allowing re-use of the same access code. In the following example, a generic access and view descriptor is needed to get data from different surface weather tables.

%LET SLCTSTR = %STR(YEAR MO DAY HR WDIR WSPD);
%LET TBLNAME = %STR(SFC724338);
%LET SUBSTR = %STR(SUBSET WHERE YEAR BETWEEN 1971 AND 1991);
PROC ACCESS DBMS=DB2;
CREATE WORK.&NAME..ACCESS;
SSID = DSN;
TABLE = ADB.&TBLNAME;
ASSIGN = YES;
CREATE WORK.&NAME..VIEW;
SELECT &SLCTSTR;
&SUBSTR;
RUN;

&NAME - holds the name of the access and view descriptors to be created. Used in both sections of code.
&SLCTSTR - holds the variables that need to be pulled from the database. Used in the view descriptor.
&SUBSTR - holds the subset delimiter. If no subset is desired, this should be empty.
&TBLNAME - holds the name of the database table to be accessed. Used in the access descriptor.

This code could be used to get data from any table, and is about as generic as it can get. The macro variables will need to be set before it is called, but such code is very useful. Ideally, the entire snipit should be made into a macro that is called whenever the database needs to be accessed for data.

THE SQL PASS-THROUGH FACILITY

There are five components that make up the Pass-Through Facility. They are briefly described below, and followed by examples of how they are used.

CONNECT Statement - establishes a connection to the relational database system.
DISCONNECT Statement - cuts the link forged with the Connect statement.
EXECUTE Statement - sends SQL statements to the database.
CONNECTION TO component - allows SAS to use the database information within an SQL query.
Return Codes - two macro variables &SQLXRC and &SQLXMSG that hold the return code and error message from the database system.

The example below creates a SAS dataset called SCOTTWIND, which contains all year, month, day, hour, wind speed and direction variables from the DB2 database for Scott AFB surface weather observations, between 1971 and 1991.

```
PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE SCOTIWND AS
(SELECT YEAR, MO, DAY, HR, WSPD, WDIR
FROM ADB.SFC72438
WHERE YEAR BETWEEN 1971 AND 1991);
%PUT &SQLXRC &SQLXMSG;
DISCONNECT FROM DB2;
QUIT;
```

The output would look like this:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MO</th>
<th>DAY</th>
<th>HR</th>
<th>WSPD</th>
<th>WDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>125</td>
</tr>
<tr>
<td>1971</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>120</td>
</tr>
</tbody>
</table>

As you can see, it's nearly standard SQL, with some extra lines added. Here's a breakdown of what is different:

CONNECT TO DB2(SSID=DSN); tells SAS to connect to the relational database system. SSID is the SubSystem ID. If left blank, it will default to whatever value is in the SAS system option DB2SSID = is set.

SELECT * FROM CONNECTION TO DB2 gets the information from the connection that has been created. Either the catch-all asterisk can be used, which returns the column names from DB2 as they exist in 082, or new variable names can be used by specifying them with the AS command: SELECT YEAR, MO AS MONTH, DAY, HR AS HOUR, WSPD AS WINDSPD, WDIR AS WINDIR FROM CONNECTION TO DB2. When using the asterisk, SAS truncates any DB2 variable name that is larger than eight characters to eight. Thus, if we had asked for ELEVATION as well, it would become ELEVATJO in our SAS dataset.

(SELECT YEAR, MO, DAY, HR, WSPD, WDIR FROM ADB.SFC72438 WHERE YEAR BETWEEN 1971 AND 1991); is a standard SQL select, being passed to DB2. The FROM clause specifies the DB2 table by name.

%PUT &SQLXRC &SQLXMSG; These two macro variables contain the return codes from the relational database. It's wise to include this line in your code for debugging reasons.

DISCONNECT FROM DB2; sever the connection with the database.

The next example shows how to create a new DB2 table, using the EXECUTE command, that will contain precipitation data.

Again, it'll be for Scott AFB, IL, and will hold year, month, day, hour, precipitation amount in tenths, and precipitation type.

```
PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
EXECUTE (CREATE TABLE ADB.SCOTT_P precip
(YEAR NUMERIC, MO NUMERIC, DAY NUMERIC, HR NUMERIC,
PRCPTYPE CHAR(5))) BY DB2;
%PUT &SQLXRC &SQLXMSG;
EXECUTE (COMMIT WORK) BY DB2;
EXECUTE (INSERT INTO ADB.SCOTT_P precip
VALUES(1993,01,01,00, 5, 'SNOW')) BY DB2;
EXECUTE (INSERT INTO ADB.SCOTT_P precip
VALUES(1993,01,03,20, 'SNOW')) BY DB2;
EXECUTE (INSERT INTO ADB.SCOTT_P precip
VALUES(1993,01,06, 00, 'NA')) BY DB2;
%PUT &SQLXRC &SQLXMSG;
EXECUTE (COMMIT WORK) BY DB2;
DISCONNECT FROM DB2;
QUIT;
```

The EXECUTE(...) BY DB2 command passes the SQL command to the database, telling it to create the new ADB.SCOTT_P precip table, and then load it with the data. Probably not the best way to load a lot of data, but it works.

**SUMMARY FUNCTIONS**

SQL can perform many of the same functions that PROC MEANS can. These functions include:

- **AVG, MEAN** - finds the average/mean of values
- **COUNT, FREQ, N** - finds the number of non-missing values
- **CSS** - finds the corrected sum of squares
- **CV** - finds the coefficient of variation in percent
- **MAX** - finds the largest value
- **MIN** - finds the smallest value
- **NMISS** - finds the number of missing values
- **PRT** - finds the probability of a greater absolute value of Student's t
- **RANGE** - finds the range of values
- **STD** - finds the standard deviation
- **STDERR** - finds the standard error of the mean
- **SUM** - finds the sum of the values
- **T** - finds Student's t value
- **USS** - finds the uncorrected sum of the squares
- **VAR** - finds the variance

These functions work against the entire column. If you wanted the total number of observations in a table, you'd use SELECT COUNT(*). To get the maximum value, you'd use MAX(column-name). DB2 returns the value name of EXPRESSN to SAS when using the summary functions. If multiple summaries are used, you'll get EXPRESSN, EXPRESS0, EXPRESS1, etc. Use the AS command to rename them to something more useful.

We're interested in finding out what the average wind speed and direction is for Scott AFB. So we use the following code, and specify what to call the averages returned:
PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE SCOTTAVG AS
SELECT EXPRESSN AS AVGWSPD, EXPRESSO AS AVGWDIR
FROM CONNECTION TO DB2
(SELECT AVG(WSPD), AVG(WDIR)
FROM ADB.SFC724338);
%PUT &SQL.XRC &SQL.XMSG;
DISCONNECT FROM DB2;
QUIT;

The output looks like this with one observation with two variables:

<table>
<thead>
<tr>
<th>AVGWSPD</th>
<th>AVGWDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>128</td>
</tr>
</tbody>
</table>

But, what if we wanted to know the monthly average by year for these two?

PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE SCOTTAVG AS
SELECT YEAR, MO, EXPRESSN AS AVGWSPD,
EXPRESSO AS AVGWDIR
FROM CONNECTION TO DB2
(SELECT YEAR, MO, AVG(WSPD), AVG(WDIR)
FROM ADB.SFC724338
GROUP BY YEAR, MO);
%PUT &SQL.XRC &SQL.XMSG;
DISCONNECT FROM DB2;
QUIT;

Here, the GROUP BY clause is used to specify that we want the averages grouped by each value in our YEAR and MO columns.

The output would look like this:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MO</th>
<th>AVGWSPD</th>
<th>AVGWDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1</td>
<td>6</td>
<td>120</td>
</tr>
<tr>
<td>1971</td>
<td>2</td>
<td>10</td>
<td>117</td>
</tr>
<tr>
<td>1971</td>
<td>3</td>
<td>12</td>
<td>119</td>
</tr>
</tbody>
</table>

MACRO VARIABLES

Macro variables can be used anywhere within the SQL code. SQL can also place values into macro variables for you.

The following example shows how you can make your SQL code more flexible by using macro variables within it.

%LET DATATBL = %STR(SCOTTWND);
%LET SLCTSTR = %STR(YEAR, MO, DAY, HR, WDIR, WSPD);
%LET TBLNAME = %STR(ADB.SFC724338);
%LET WHERESTR = %STR(WHERE YEAR BETWEEN 1971 AND 1991);

PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE &DATATBL AS
(SELECT &SLCTSTR
FROM &TBLNAME
WHERE &WHERESTR);
%PUT &SQL.XRC &SQL.XMSG;
DISCONNECT FROM DB2;
QUIT;

This same piece of code can be used with different tables and variables by simply changing the macro variable assignments. Note that &WHERESTR could be left blank, thus returning all of the data in the DB2 table with no constraints.

In order to find the minimum and maximum winds for Scott AFB, and the days on which they occurred, we could use the next example. Two SQL statements will be used to do this; the first uses the MIN and MAX summary functions, and placed them into &MINWIND and &MAXWIND. The second will return the year, month, day and hour of whenever the minimum and maximum winds occur.

PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
SELECT MIN, MAX INTO :MINWIND, :MAXWIND
FROM ADB.SFC724338;
%PUT &SQL.XRC &SQL.XMSG;
DISCONNECT FROM DB2;
QUIT;

PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE MINMAX AS
(SELECT YEAR, MO, DAY, HR, WSPD, WDIR
FROM ADB.SFC724338
WHERE WSPD = &MINWND OR WSPD = &MAXWND);
%PUT &SQL.XRC &SQL.XMSG;
DISCONNECT FROM DB2;
QUIT;

The output from the first would be &MINWIND holding 0, and &MAXWIND holding 49.

Output from the second would be all observations where the winds were either 0 or 49.

Note that in the first, you must specify what values you are getting from DB2, instead of using the catch-all asterisk. In SAS/SQL, the INTO command will place the values returned into macro variables, whose names must be preceded with a colon, i.e.: :MAXWND. Also, notice the difference in how the INTO works, as opposed to the AS. Here we list all of the variables, followed by the INTO, and then the names of the macro variables we want the values put into.

In the second SQL code, a dataset called MINMAX is created that contains the dates, wind speeds and directions of any observations that match our min and max.

SELECTING FROM MULTIPLE TABLES

Often you will need to get information from two or more different tables. This can easily be accomplished with SQL. In the following code, we will get the forecast wind speed and direction from ADB.TAF724338, and what actually was observed for that hour from the ADB.SFC724338 table.

PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE &DATATBL AS
SELECT * FROM CONNECTION TO DB2
(SELECT &SLCTSTR
FROM &TBLNAME
WHERE &WHERESTR);
%PUT &SQL.XRC &SQL.XMSG;
DISCONNECT FROM DB2;
QUIT;
PROC SQL NOPRINT:
CONNECT TO DB2(SSID=DSN):
CREATE TABLE SCOTI2 AS
SELECT YEAR, MO, DAY, HR, TAFWDIR,
WDIR AS OBSWDIR
FROM CONNECTION TO DB2
(SELECT A.YEAR, AMO, A.DAY, A.HR,
A.TAFWDIR,
B.WDIR
FROM ADB.TAF724338 AS A,
ADB.SFC724338 AS B
WHERE A.YEAR = B.YEAR
AND A.MO = B.MO
AND A.DAY = B.DAY
AND A.HR = B.HR):
%PUT &SQLXRC &SQLXMSG:
DISCONNECT FROM DB2:
QUIT:

Output looks like this:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MO</th>
<th>DAY</th>
<th>HR</th>
<th>TAFWDIR</th>
<th>OBSWDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>120</td>
<td>120</td>
</tr>
</tbody>
</table>

Once again, the names of the variables are specified in the first SELECT statement. Here we have to because we're getting two different WSPD and WDIR values. Our first SELECT statement will plug those returned values into the new names TAFWSPD, TAFWDIR, OBWSWSPD, OBWSWDIR.

In the code being passed to DB2, we use table aliases to specify the columns we are getting from each of the two DB2 tables. The line FROM ADB.TAF724338 AS A, ADB.SFC724338 AS B sets up the aliases using the AS operator. Our passed SELECT statement tells SQL that we want year, month, day, hour, wspd and wdir from the ADB.TAF724338 table, and wdir and wspd from the ADB.SFC724338 table, where the dates match up in the two. Please note, that you'll only get data back when the dates match...you won't get any data from either table when the dates DON'T match up.

A multi-table select can be across many tables, but, the more tables you query, the longer the SQL takes to run.

**THE CASE EXPRESSION**

The case expression is used to select result values if certain conditions are met. It can be used to categorize values. The format of the case expression is:

```
CASE <case-operand>
  WHEN <when-condition> THEN <result-expression>
  WHEN <when-condition> THEN <result-expression>
  <ELSE result-expression>
END
```

The WHEN-THEN clauses are used to evaluate the condition and give the result action for that condition. The option ELSE is the catch-all for anything that doesn't meet your tests. When the case-operand is specified, it is compared to the when-condition for equality. If they are equal, then the WHEN clause is true and the result-expression occurs. If the case-operand isn't specified after the CASE, it must be specified in the WHEN filed.

```
WHEN 1 THEN 'YES'
WHEN 0 THEN 'NO'
ELSE 'MAYBE'
END
```

```
WHEN TEST=1 THEN 'YES'
WHEN TEST=2 THEN 'NO'
ELSE 'MAYBE'
END
```

Let's say we want to categorize surface winds at Scott AFB into several groups; Calm, Mild, Strong, and Severe. The case expression lets us do this without an extra SAS data step:

```
PROC SQL NOPRINT:
CONNECT TO DB2(SSID=DSN):
CREATE TABLE WINDGRP AS
SELECT YEAR, MO, DAY, HR, WDIR, WSPD,
CASE
  WHEN WSPD < 2 THEN 'CALM'
  WHEN WSPD < 15 THEN 'MILD'
  WHEN WSPD < 55 THEN 'STRONG'
  ELSE 'SEVERE'
END AS CATEGORY
FROM CONNECTION TO DB2
(SELECT YEAR, MO, DAY, HR, WDIR, WSPD,
FROM ADB.SFC724338):
%PUT &SQLXRC &SQLXMSG:
DISCONNECT FROM DB2:
QUIT:
```

Note that we had to put the CASE statement in the local SELECT, and not in the DB2 SELECT. Also, this shows an example of how an expression can be renamed just like a column/variable name, using the AS command. This piece of code will give us a dataset with year, mo, day, hour, wdir, wspd and category.

Output would look like this:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MO</th>
<th>DAY</th>
<th>HR</th>
<th>WSPD</th>
<th>WDIR</th>
<th>CAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>125</td>
<td>MILD</td>
</tr>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>120</td>
<td>CALM</td>
</tr>
<tr>
<td>1971</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>30</td>
<td>190</td>
<td>STRONG</td>
</tr>
</tbody>
</table>

**DIFFERENCES BETWEEN ACCESS AND SQL**

Both methods allow easy access to data from databases. However, ACCESS calls are converted to SQL in order to access the database, and thus the more complex the subset clauses, the slower ACCESS is going to run compared to SQL. SQL, via the pass through facility, gives its queries directly to the database, further speeding up the process. You'll only see major time differences on very large datasets, and on complex queries with numerous subsetting clauses. SQL has the added advantage of being able to directly update database tables.
REFERENCES


SAS and SAS/ACCESS are registered trademarks of SAS Institute, Inc. in the USA and other countries.

DB2 is a registered trademark of IBM in the USA and other countries.

CONTACTING THE AUTHOR

C. Michael Whitney, SSgt, USAF
USAFETAC / SYS
Bldg. 859
Scott AFB, IL 62225-5116
(618) 256-5323
Internet: syscmw@lightning.safb.af.mil