Paper 143-2009

Don't Be a SAS® Dinosaur: Modernizing Programs with Base SAS 9.2 Enhancements

Warren Repole Jr., SAS Institute Inc.

ABSTRACT

New features of the SAS® programming language often permit the replacement of complex algorithms and clunky workarounds with more elegant code. Some enhancements support the creation of more efficient solutions. Whether you are a SAS programmer with many years of experience or a novice user who is responsible for maintaining legacy programs, implementing updated approaches can allow you to streamline your SAS applications, expedite the development and debugging process, and minimize future maintenance of the code.

INTRODUCTION

This paper covers a selection of enhancements implemented in SAS 9.2, including new features of Base SAS procedures, the DATA step, functions, formats, and the macro language. Each enhancement is paired with an alternative technique available prior to SAS 9.2. This allows for comparison of older and newer approaches.

The primary motivation for exploiting SAS 9.2 enhancements is the ability to create robust yet easy-to-maintain SAS programs with less programmer effort. Efficiency gains in terms of machine resources might be achieved, but they are of secondary importance in most examples.

BASE SAS PROCEDURES

PROC PRINT: BLANKLINE= OPTION

Inserting a blank line into PROC PRINT output after every *n* observations could enhance readability, but this was difficult to accomplish prior to SAS 9.2. Observations with missing values were inserted, the MISSING= system option was switched to a blank (then reset to its default value afterward), and the NOOBS option was required.

In SAS 9.2 producing this type of report is simple using the BLANKLINE= option in the PROC PRINT statement.

```
data class_blanks(drop=i);
  set sashelp.class;
  output;
  if mod(_n_,5)=0;
  array alln {*} _numeric_ ;
  array allc {*} _character_ ;
  do i=1 to dim(alln); alln{i}=.;   end;
  do i=1 to dim(allc); allc{i}=" "; end;
  output;
run;
options missing=" ";
proc print data=class_blanks  noobs ;
run;
options missing=".";
```

Figure 1a.
PROC PRINT code before SAS 9.2

```
proc print data=sashelp.class
           blankline=5 ;
run;
```

Figure 1b.
PROC PRINT code with BLANKLINE= option

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	56.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
18	Thomas	M	11	57.5	85.0
19	William	M	15	66.5	112.0

Figure 1c. Output from Figure 1b program including the Obs column suppressed in Figure 1a program

PROC SORT: SORTSEQ= OPTION

Sorting character data that is stored inconsistently in terms of case has been a challenge for many years. Typical solutions involved converting the data to uppercase using the UPCASE function before sorting.

In SAS 9.2 the SORTSEQ= value of LINGUISTIC with the collating rule STRENGTH=PRIMARY supports sorting that is not case sensitive. Diacritical differences and punctuation are handled through other STRENGTH= settings.

```
data french;
  set maps.names;
  where Territory contains "France";
  Territory_Upper=upcase(Territory);
run;
proc sort data=french;
  by Territory_Upper  Name ;
run;
```

Figure 2a.
PROC SORT using uppercase values

```
proc sort data=maps.names out=french
          sortseq=linguistic
                    (strength=primary) ;
  where Territory contains "France";
  by Territory  Name ;
run;
```

Figure 2b.
PROC SORT using SORTSEQ=LINGUISTIC option

Collating rules associated with the sorted data are exploited in subsequent BY statements.

```
proc print data=french;
  by Territory;
  id Territory;
  var Territory Name;
run;
```

TERRITORY	TERRITORY	NAME
Overseas Department of France	Overseas Department of France	FRENCH GUIANA
	Overseas Department of France	GUADELOUPE
	Overseas Department of France	MARTINIQUE
Overseas territory of France	Overseas territory of France	FRENCH POLYNESIA
	overseas territory of France	MAYOTTE
	Overseas territory of France	NEW CALEDONIA
	Overseas territory of France	ST. PIERRE/MIQUELON
	Overseas territory of France	TOGO
	Overseas territory of France	WALLIS/FUTUNA ISLANDS

Figure 2c. Program and output illustrating the inconsistent case of Territory in the MAYOTTE observation

The collating rule ALTERNATE_HANDLING=SHIFTED treats spaces and punctuation as minimally important.

City New Castle New Market Newington Newport News Newsoms **Figure 3a.** **Data order from PROC SORT** **without ALTERNATE_HANDLING=SHIFTED**	City New Castle Newington New Market Newport News Newsoms **Figure 3b.** **Data order from PROC SORT** **with ALTERNATE_HANDLING=SHIFTED**

The collating rule NUMERIC_COLLATION=ON handles integer values within a string as their numeric equivalent for sorting purposes.

Address 10274 Dinwiddie Ct 133 Dinwiddie Ct 9658 Dinwiddie Ct **Figure 4a.** **Data order from PROC SORT** **without NUMERIC_COLLATION=ON**	Address 133 Dinwiddie Ct 9658 Dinwiddie Ct 10274 Dinwiddie Ct **Figure 4b.** **Data order from PROC SORT** **with NUMERIC_COLLATION=ON**

DATA STEP TECHNIQUES

SET STATEMENT: DATA SET LISTS AND INDSNAME= OPTION

Providing a list of commonly named data sets in the SET (or MERGE) statement involved either hardcoding the data set names or implementing a macro-based solution. Identifying the source data set for each observation required the use of the IN= data set option and significant conditional logic.

In SAS 9.2 data set lists using the dash or colon are supported. Data set options to be applied to all input data sets can be specified just once. The INDSNAME= option in the SET statement, also new in SAS 9.2, provides a temporary variable that contains the fully qualified name of the contributing data set.

```
data combined;
  set sashelp.prdsale(in=in_prdsale
         keep=country product year actual)
      sashelp.prdsal2(in=in_prdsal2
         keep=country product year actual)
      sashelp.prdsal3(in=in_prdsal3
         keep=country product year actual)
  ;
  length Source $ 32;
  if in_prdsale then Source="PRDSALE";
  else
    if in_prdsal2 then Source="PRDSAL2";
  else
    if in_prdsal3 then Source="PRDSAL3";
run;
```

Figure 5a.
DATA step with hardcoded data set names, repeated data set options, and conditional logic

```
data combined;
  set sashelp.prdsal:
        (keep=country product year actual)
      indsname=inputdsn
  ;
  length Source $ 32;
  Source=scan(inputdsn,2);
run;
```

Figure 5b.
DATA step with data set name list, single data set option specification, and INDSNAME= option

```
proc freq data=combined;
  tables Source;
run;
```

Source	Frequency	Percent	Cumulative Frequency	Cumulative Percent
PRDSAL2	23040	64.00	23040	64.00
PRDSAL3	11520	32.00	34560	96.00
PRDSALE	1440	4.00	36000	100.00

Figure 5c. Program and output illustrating the distribution of input observations

INFILE AND FILE STATEMENTS: DLMSTR= AND DLMSOPT= OPTIONS

Parsing text input lines using the DLM= option in the INFILE statement is limited to using one or more single characters as field delimiters. To treat a character string as a delimiter, additional DATA step logic was required.

In SAS 9.2 the DLMSTR= option provides a mechanism to use a string as a delimiter. The case of the delimiter string can be ignored by adding the DLMSOPT="I" option.

```
data stringdelim;
  infile datalines;
  input;
  _infile_=tranwrd(_infile_,"{sep}","/");
  N1=input(scan(_infile_,1,"/"),32.);
  Length N2 $ 8;
  N2=scan(_infile_,2,"/");
  N3=input(scan(_infile_,3,"/"),32.);
datalines;
123{sep}sep{sep}789
0{sep}ABCXYZ{sep}456
run;
```

Figure 6a.
Parsing multiple character delimiters using DATA step functions

```
data stringdelim;
  infile datalines dlmstr="{sep}"
                    dlmsopt="I";
  input N1 N2 $ N3;
datalines;
123{sep}sep{sep}789
0{Sep}ABCXYZ{SEP}456
run;
```

Figure 6b.
Parsing multiple character delimiters using the DLMSTR= option

```
proc print data=stringdelim;
run;
```

Obs	N1	N2	N3
1	123	sep	789
2	0	ABCXYZ	456

Figure 6c. Program and output illustrating the success of the delimiter identification

DATA STATEMENT: NOLIST OPTION

When data errors are encountered in a DATA step, such as invalid raw data values for numeric variables and unsuccessful observation retrieval using the KEY= option in the SET statement, the automatic _ERROR_ variable is set to 1, triggering a "dump" of the Program Data Vector (PDV) that lists all variables available in the DATA step.

Setting the system option ERRORS=0 (changing from its typical default value of 20) eliminates the PDV dump but also suppresses potentially valuable associated messages.

The PDV dump can also be suppressed by resetting _ERROR_ to 0 prior to the end of the DATA step iteration. The associated messages still appear but are no longer restricted by the ERRORS= system option value, introducing the possibility for a massive volume of messages.

In SAS 9.2 the NOLIST option at the end of the DATA statement suppresses the PDV dump while generating associated messages consistent with the current ERRORS= system option setting. A single-line note appears in the SAS log in place of the complete PDV dump, a reasonable compromise outcome.

```
data nopdvdump ;
  input N1 N2;
  _error_=0;
datalines;
123 234
234 xyz
abc 345
run;
```

Figure 7a.
Suppressing a PDV dump by setting _ERROR_ to 0

```
data nopdvdump92 / nolist ;
  input N1 N2;
datalines;
123 234
234 xyz
abc 345
run;
```

Figure 7b.
Suppressing a PDV dump by using the NOLIST option

```
17   data nopdvdump92 / nolist ;
18     input N1 N2;
19   datalines;

NOTE: Invalid data for N2 in line 21 5-7.
NOTE: NOLIST option on the DATA statement suppressed output of variable listing.
NOTE: Invalid data for N1 in line 22 1-3.
NOTE: NOLIST option on the DATA statement suppressed output of variable listing.
NOTE: The data set WORK.NOPDVDUMP92 has 3 observations and 2 variables.
```

Figure 7c. SAS log excerpt illustrating the suppression of the PDV dump when invalid data is encountered

SAS FUNCTIONS

COUNTW AND FINDW FUNCTIONS

Parsing a character string often involves a conditional loop that detects the end of the string. Locating a substring is challenging when inadvertent matches must be avoided and substrings could appear at the start, the end, or both.

In SAS 9.2 the COUNTW function determines how many words are contained in a string, using a similar algorithm to the SCAN function.

```
data words;
  set sashelp.zipcode;
  Words=0;
  do while(scan(City,Words+1) ne " ");
    Words+1;
  end;
run;
```

Figure 8a.
DO WHILE to count words in a string

```
data words;
  set sashelp.zipcode;
  Words=countw(city);
run;
```

Figure 8b.
COUNTW function to count words in a string

```
proc freq data=words;
  tables Words;
run;
```

Words	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	31700	74.92	31700	74.92
2	9522	22.50	41222	97.42
3	470	1.11	41692	98.53
4	258	0.61	41950	99.14
5	361	0.85	42311	100.00
6	1	0.00	42312	100.00

Figure 8c. Program and output illustrating the distribution of the Words variable

In SAS 9.2 the FINDW function locates a substring that exists as a separate word within a longer string. The FINDW function can return either the character position at which the substring was found, or the word number corresponding to the substring match (using the E modifier). The I modifier requests that the search process should ignore case.

```
data lake;
  set sashelp.zipcode;
  LakePos=index(" "||upcase(City)||" ",
                " LAKE ");
  if LakePos > 0
  then do LakeWord=1 to 99999
     while(scan(City,LakeWord," ")
            not in  ("Lake" " "));
  end;
run;
```

Figure 9a.
INDEX and UPCASE functions to locate word
Conditional iterative DO loop to locate word number

```
data lake;
  set sashelp.zipcode;
  LakePos=findw(City,"lake"," ","i");
  if LakePos > 0
  then LakeWord=findw(City,"lake"," ",
                    "ie");
run;
```

Figure 9b.
FINDW function to locate word and word number

```
proc freq data=lake;
  tables LakePos LakeWord;
run;
```

LakePos	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	41880	98.98	41880	98.98
1	150	0.35	42030	99.33
5	7	0.02	42037	99.35
6	97	0.23	42134	99.58
7	57	0.13	42191	99.71
8	60	0.14	42251	99.86
9	30	0.07	42281	99.93
10	19	0.04	42300	99.97
11	7	0.02	42307	99.99
12	1	0.00	42308	99.99
14	1	0.00	42309	99.99
15	2	0.00	42311	100.00
16	1	0.00	42312	100.00

LakeWord	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	150	34.72	150	34.72
2	275	63.66	425	98.38
3	7	1.62	432	100.00

Frequency Missing = 41880

Figure 9c. Program and output illustrating the distribution of the LakePos and LakeWord variables

CHAR AND FIRST FUNCTIONS

The SUBSTR function is commonly used to extract text from a string. Using the value 1 for the third argument extracts a single character. Using the value 1 for both the second and third arguments extracts the first character only. If the SUBSTR function result is stored in a new variable, the length of that variable defaults to the length of the original string.

In SAS 9.2 the CHAR function extracts a single character, assigning a length of 1 to any resulting new variable. The FIRST function performs a similar action, extracting only the first character from the original string.

Figure 10a	Figure 10b

<pre>data CityPunct;
  set maps.uscity;
  where substr(City,1,1)="O";
  PunctPosition=anypunct(City);
  if PunctPosition > 0;
  PunctChar=substr(City,PunctPosition,1);
run;</pre>
	<pre>data CityPunct;
	  set maps.uscity;
	  where **first(City)**="O";
	  PunctPosition=anypunct(City);
	  if PunctPosition > 0;
	  PunctChar=**char(City,PunctPosition)**;
	run;</pre>

Figure 10a. **LENGTH statement with SUBSTR function**	**Figure 10b.** **FIRST and CHAR functions**

```
proc freq data=CityPunct;
   tables PunctChar;
run;
```

Punct Char	Frequency	Percent	Cumulative Frequency	Cumulative Percent
'	7	36.84	7	36.84
-	12	63.16	19	100.00

Figure 10c. Program and output illustrating the distribution of the PunctChar variable

```
proc contents data=CityPunct varnum;
run;
```

Output from Figure 10a program

Variables in Creation Order

#	Variable	Type	Len	Label
1	CITY	Char	80	City Name
2	PunctPosition	Num	8	
3	PunctChar	Char	80	

Output from Figure 10b program

Variables in Creation Order

#	Variable	Type	Len	Label
1	CITY	Char	80	City Name
2	PunctPosition	Num	8	
3	PunctChar	Char	1	

Figure 10d. Program and output illustrating the length of the PunctChar variable using each technique

NWKDOM AND HOLIDAY FUNCTIONS

Deriving the *n*th occurrence of a given weekday within a month has been solved through algorithms involving functions such as INTNX. Occasionally, the desired date is a holiday or observance such as Thanksgiving or Easter.

In SAS 9.2 dates for common holidays in the United States and Canada can be determined using the HOLIDAY function. The NWKDOM function is more general in nature, returning the date corresponding to the *n*th occurrence of a given weekday within a month. To obtain the final occurrence, you can use 5; if there are only 4 occurrences, then the 4th occurrence is returned.

```
data SpecialDates;
  Election2008=intnx("week.2",
                      "31oct2008"d,1)+1;
  Memorial2009=intnx("week.2",
                      "31may2009"d,0);
  if weekday("01jul2012"d)=1
  then Canada2012="02jul2012"d;
  else Canada2012="01jul2012"d;
run;
```

Figure 11a.
Date manipulation using INTNX and WEEKDAY functions

```
data SpecialDates;
  Election2008=nwkdom(1,2,11,2008)+1;
  Memorial2009=nwkdom(5,2,5,2009);
  Canada2012=holiday("CANADAOBSERVED",2012);
  Easter2014=holiday("EASTER",2014);
run;
```

Figure 11b.
HOLIDAY and NWKDOM functions

```
proc print data=SpecialDates;
   format _all_ weekdate20.;
run;
```

```
     Election2008          Memorial2009

Tue, Nov 4, 2008       Mon, May 25, 2009

       Canada2012             Easter2014

Mon, Jul 2, 2012       Sun, Apr 20, 2014
```

Figure 11c. Program and output illustrating the derived dates for the selected observances

SAS FORMATS

DATE*w*. FORMAT: WIDTH OF 11

The DATE*w*. format displays the day number, month abbreviation, and year for a SAS date. To produce the same display with dashes between the components, a custom date format can be created through PROC FORMAT.

In SAS 9.2 the DATE*w*. format allows a width of 11, automatically inserting dashes between the date components.

```
proc format;
  picture mydate (default=11)
        .="Missing date"
    other='%0d-%b-%Y' (datatype=date);
run;
proc print data=sashelp.prdsal3;
  format Date mydate.;
run;
```

Figure 12a.
Defining a custom date format with PROC FORMAT

```
proc print data=sashelp.prdsal3;
  format Date date11.;
run;
```

Figure 12b.
Using the DATE11. format

Obs	DATE
1	01-JAN-1997
2	01-FEB-1997
3	01-MAR-1997
4	01-APR-1997
5	01-MAY-1997

Figure 12c. Output illustrating the first few formatted values of the DATE variable

PERCENTN*w*. FORMAT

The PERCENT*w*. format displays a numeric value as a percentage with the percent sign (%) at the end of the value. Negative values are displayed surrounded by parentheses. To display a leading minus sign instead of parentheses, a custom format can be created through PROC FORMAT.

In SAS 9.2 a negative percentage can be displayed with a leading minus sign using the PERCENTNw. format. When using the PERCENTN*w*. format, add 1 to the anticipated maximum width to account for a trailing blank.

```
data Differences;
  set sashelp.prdsal3;
  DiffPct=(Actual-Predict)/Predict;
run;
proc format;
  picture pcntneg
     low-<0="009.9%"
               (mult=1000 prefix="-")
     0-high="009.9%" (mult=1000);
run;
proc print data=Differences;
   var Actual Predict DiffPct;
   format DiffPct pcntneg7.;
run;
```

Figure 13a.
Defining a picture format with PROC FORMAT

```
data Differences;
  set sashelp.prdsal3;
  DiffPct=(Actual-Predict)/Predict;
run;

proc print data=Differences;
   var Actual Predict DiffPct;
   format DiffPct percentn8.1;
run;
```

Figure 13b.
Using the PERCENTN*w*. format

Due to the truncation that can occur with a picture format, the results are slightly different. The PERCENTN*w*. format applies the same type of rounding algorithm as other SAS formats when displaying a limited number of digits.

ACTUAL	PREDICT	DiffPct
$726.00	$509.00	42.6%
$1,311.00	$418.00	213.6%
$24.00	$12.00	100.0%
$1,342.00	$556.00	**141.3%**
$552.00	$532.00	**3.7%**
$1,784.00	$1,786.00	-0.1%
$1,317.00	$1,655.00	-20.4%
$1,678.00	$258.00	**550.3%**
$1,852.00	$1,277.00	45.0%
$1,056.00	$1,594.00	**-33.7%**

Figure 13c.
Output from Figure 13a program

ACTUAL	PREDICT	DiffPct
$726.00	$509.00	42.6%
$1,311.00	$418.00	213.6%
$24.00	$12.00	100.0%
$1,342.00	$556.00	**141.4%**
$552.00	$532.00	**3.8%**
$1,784.00	$1,786.00	-0.1%
$1,317.00	$1,655.00	-20.4%
$1,678.00	$258.00	**550.4%**
$1,852.00	$1,277.00	45.0%
$1,056.00	$1,594.00	**-33.8%**

Figure 13d.
Output from Figure 13b program

SAS MACRO LANGUAGE

IN OPERATOR: MINOPERATOR AND MINDELIMITER= OPTIONS

Expressions evaluated by the %IF statement have been limited to pairwise comparisons. To check one value against a list of values, a compound expression or a function-based solution was required.

In SAS 9.2 the IN operator is supported with %IF statements when the MINOPERATOR option is added to the %MACRO statement or set as a SAS system option. The macro version of the IN operator uses spaces as default delimiters, so the MINDELIMITER= option is available to change the delimiter to another character.

```
%macro Filter(ageparm);

  %if &ageparm=11 or &ageparm=12 or
      &ageparm=13 or &ageparm=14 or
      &ageparm=15 or &ageparm=16
  %then %do;
    proc print data=sashelp.class;
      where age = &ageparm;
      title1 "Students of Age &ageparm";
    run;
  %end;
  %else %do;
     %put ERROR: No matching students.;
     %put ERROR- Valid ages are;
     %put ERROR- 11 12 13 14 15 16;
  %end;

%mend Filter;
```

Figure 14a.
Defining a compound %IF expression

```
%macro Filter(ageparm) / minoperator;

  %if &ageparm in 11 12 13 14 15 16
  %then %do;
    proc print data=sashelp.class;
      where age = &ageparm;
      title1 "Students of Age &ageparm";
    run;
  %end;
  %else %do;
     %put ERROR: No matching students.;
     %put ERROR- Valid ages are;
     %put ERROR- 11 12 13 14 15 16;
  %end;

%mend Filter;
```

Figure 14b.
Using the IN operator in the %IF expression

```
1115  %Filter(13)
MPRINT(FILTER):   proc print data=sashelp.class;
MPRINT(FILTER):   where age = 13;
MPRINT(FILTER):   title1 "Students of Age 13";
MPRINT(FILTER):   run;
NOTE: There were 3 observations read from the data set SASHELP.CLASS.
      WHERE age=13;

1116  %Filter(17)
ERROR: No matching students.
       Valid ages are
       11 12 13 14 15 16
```

Figure 14c. SAS log excerpt illustrating the conditional logic results of the %IF expressions

CONCLUSION

Each release of SAS software contains valuable new features that can enhance the programming experience. Some changes permit more reliable implementation of common tasks while others can make difficult tasks considerably easier, more efficient, or both.

A brief glance at the "What's New" documentation is a worthwhile investment for any SAS programmer, regardless of their level of experience.

REFERENCES

- Olson, Diane. 2008. "New in SAS® 9.2: It's the Little Things That Count." Proceedings of the *SAS Global Forum 2008 Conference*. Cary, NC: SAS Institute Inc. Available at www2.sas.com/proceedings/forum2008/176-2008.pdf
- SAS Institute Inc. 2008a. *Base SAS® 9.2 Procedures Guide*. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 2008b. *SAS® 9.2 Language Reference: Dictionary*. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 2008c. *SAS® 9.2 Macro Language: Reference*. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 2008d. *What's New in SAS® 9.2*. Cary, NC: SAS Institute Inc.

- Secosky, Jason. 2008. SAS Institute Inc., Cary, NC. "A Sampler of What's New in Base SAS® 9.2". Available at support.sas.com/rnd/base/datastep/whats-new-base-sas92.pdf

ACKNOWLEDGMENTS

The author acknowledges his fellow instructors in the SAS Education Division, especially Jim Simon, for reviewing this paper and providing helpful feedback.

The concept for this paper was initially presented during the seminar "Modernizing Your SAS® Code, or How to Avoid Becoming a SAS Dinosaur" at SAS Global Forum 2007 in Orlando FL, adapted from the author's publication *Don't Be a SAS® Dinosaur: Modernize Your SAS Code*.

RECOMMENDED READING

- The author has contributed several SAS Notes related to topics discussed in this paper.

  SAS Institute Inc. 2008. SAS Note 31366, “Inserting a Blank Line After Every N Observations in PROC PRINT Output Using SAS 9.2.” Available at support.sas.com/kb/31/366.html

  SAS Institute Inc. 2008. SAS Note 31369, “Sorting Text Without Regard to Case in SAS 9.2.” Available at support.sas.com/kb/31/369.html

- Visit www.repole.com/dinosaur for additional examples related to the "SAS Dinosaur" concept.
- The author also maintains a presence on the **sasCommunity.org** Web site. Visit www.sascommunity.org/wiki/User:Sasdinosaur for more information.

CONTACT INFORMATION

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

Warren Repole Jr.
SAS Institute Inc.
1705 Palm Springs Dr.
Vienna VA 22182
703-255-5476
Warren.Repole@sas.com
sasdinosaur@repole.com