New Functions in SAS®9 – A Sampling

Keith Cranford, Office of the Attorney General of Texas, Child Support Division

Abstract

SAS®9 provides many new functions, including many new character functions, a few new descriptive statistics functions, and many other miscellaneous functions. This paper gives an overview for many of these functions, including syntax and examples.

Character Functions

SAS®9 greatly expands the list of functions available for character manipulation. These include functions that search for types of characters, join two or more strings together, compare strings, and much more. These new functions will be presented in groups as they relate to one another. In most cases, the function syntax will be shown accompanied by an example illustrating use of the function. However, due to the large number of functions, sometimes only the name and a description are given, especially when there are a number of functions that work similarly.

Search Functions

Searching capabilities have been expanded through a series of functions that either checks for the existence of a type of character (preceded by ANY) or that a certain type is not contained in the string (preceded by NOT). These are:

ANYALPHA	NOTALPHA
ANYALNUM	NOTALNUM
ANYDIGIT	NOTDIGIT
ANYPUNCT	NOTPUNCT
ANYSPACE	NOTSPACE
ANYXDIGIT	NOTXDIGIT
ANYPRINT	NOTPRINT
ANYGRAPH	NOTGRAPH
ANYNAME	NOTNAME
ANYFIRST	NOTFIRST
ANYCNTRL	NOTCNTRL
ANYUPPER	NOTUPPER
ANYLOWER	NOTLOWER

Syntax:

ANYALPHA(*string* <,*start*>)
NOTALPHA(*string* <,*start*>)
etc.

Each of these functions takes a character expression or variable as its first argument and a second optional argument indicates at which position to start. The result is either the position of the first character that meets the specified type or a value of zero if the type of character is not present. ANYALPHA searches for any alphabetic character, ANYALNUM searches for any alphanumeric character, ANYDIGIT for any numeric character, ANYPUNCT for punctuation, and ANYSPACE for any white space character (space, tab, line feed, etc.). Similarly, NOTALNUM searches for any character that is not alphanumeric, NOTALPHA for those not alphabetic, NOTDIGIT for those not numeric, and NOTUPPER for those not upper case alphabetic.

Example 1

```
select string,
       anyalnum(string) as AlphaNum,
       anyalpha(string) as Alpha,
       anydigit(string) as Num,
       anypunct(string) as Punctuation,
       anyspace(string) as Space
from test ;

select string,
       notalnum(string) as Not_AlphaNum,
       notalpha(string) as Not_Alpha,
       notdigit(string) as Not_Num,
       notupper(string) as Not_Upper
from test ;
```

This results in the following output.

string	AlphaNum	Alpha	Num	Punctuation	Space
abc	1	1	0	0	4
123	1	0	1	0	4
abc123	1	1	4	0	7
ABC/123	1	1	5	4	8
abc 123	1	1	5	0	4

and

string	Not_AlphaNum	Not_Alpha	Not_Num	Not_Upper
abc	4	4	1	1
123	4	1	4	1
abc123	7	4	1	1
ABC/123	4	4	1	4
abc 123	4	4	1	1

Note that all the strings begin with alphanumeric characters, but '123' does not contain any alphabetic characters (hence the zero for Alpha). There are no numeric characters in 'abc' and only 'ABC/123' contains punctuation. All the strings have spaces either imbedded or at the end.

All of the strings have at least one non-alphanumeric (space is not considered alphanumeric). '123' begins with a non-alpha character, but the others do not have a non-alpha character until the fourth character, which is a space, number, or punctuation. All but '123' begin with a non-numeric character, whereas the space accounts for the first non-numeric character in '123'. Finally, 'ABC/123' does not have a non-upper case character until the fourth position, while all others begin with non-upper case characters. The other listed functions worked similarly.

Two new functions, FIND and FINDC, search for either a specific substring or specific characters in a string. They have additional options to modify the substring such as ignoring case and to specify a position to begin the search.

Syntax:

FIND(*string,substring<,modifiers><,startpos>***)**
FINDC(*string,characters<,modifiers><,startpos>***)**

Example 2

```
select name,
       find(name,'Sam','i') as FindSam_nocase,
       find(name,'Sam',5) as FindSam_start5,
       findc(name,'st','i') as Findc_ST
from test ;
```

results in

name	FindSam_ nocase	FindSam_ start5	Findc_ST
BOB JONES	0	0	9
JaNe dOe	0	0	0
SAM SPADE	1	0	1
MEL O'DAY	0	0	0
sally ride	0	0	1

The first FIND discovers 'Sam' in the first position for 'SAM SPADE', ignoring case. The other modifier is 't' to trim trailing blanks in the string and substring. The second FIND does not find any 'Sam' starting the search in the fifth position. The FINDC function searches for occurrences of 's' or 't' ignoring case. This occurs in the ninth position in 'BOB JONES' and in the first position in 'SAM SPADE' and 'sally ride'.

Concatenation

There is a new series of concatenation functions and call routines. These function and routines will generally be faster than using other functions or the concatenation operator. These handle blanks in various ways or insert separators. Each of the functions, except for CAT, has a corresponding call routine which works in a similar manner. The arguments for the functions are lists of strings to be concatenated, with CATX having an additional argument of the separator character. Although the following example shows concatenating two strings, the functions can concatenate any number of strings.

Syntax:

CAT(*string-1* <, ... *string-n*>**)**
CATS(*string-1* <, ... *string-n*>**)**
CATT(*string-1* <, ... *string-n*>**)**
CATX(*separator*, *string-1* <, ...*string-n*>**)**
CALL CATS(*result* <, *string-1*, ...*string-n*>**);**
CALL CATT(*result* <, *string-1*, ...*string-n*>**);**
CALL CATX(*separator*, *result*<, *string-1* , ...>*string-n***)**;

Example 3

```
select string1, string2,
       cat(string1, string2) as CAT,
       cats(string1, string2) as CATS,
       catt(string1, string2) as CATT,
       catx(' ', string1, string2) as CATX
from test ;
```

results in

string1	string2	CAT	CATS	CATT	CATX
abc	123	abc123	abc123	abc123	abc 123
ab	23	ab 23	ab23	ab 23	ab 23
bc	12	bc12	bc12	bc12	bc 12
abc	23	abc 23	abc23	abc 23	abc 23
ab	12	ab 12	ab12	ab12	ab 12

CAT concatenates strings without removing blanks. Notice the leading blanks as well as blanks in the middle of the resulting strings. CATS concatenates and removes leading and trailing blanks. CATT removes only trailing blanks. Notice the leading blank in ‘ bc12’, and the blank in ‘ab 23’ is from the leading blank in ‘ 23’. Lastly, CATX removes leading and trailing blanks, but allows you to specify a separator character. In this example, a space is used for the separator.

Trimming Blanks

The STRIP function removes both leading and trailing blanks from a string. This can be used in place of the combination of TRIM and LEFT.

Syntax

STRIP(*string***)**

Example 4

```
select string1, string2,
       strip(string1)||strip(string2) as strip
from test ;
```

results in

string1	string2	strip
abc	123	abc123
ab	23	ab23
bc	12	bc12
abc	23	abc23
ab	12	ab12

Of course, the result here is the same as the previous example of the CATS function.

String Comparisons

There are several new functions used for comparing strings. The COMPARE function returns the position of the left-most character by which two strings differ, or zero if there is no difference. There is an optional argument that modifies the comparison. The modifiers specify whether to trim leading blanks, ignore case, or truncate the comparison to the shorter of the two strings. Also, the functions COMPGED and COMPLEV and the routine CALL COMPCOST compare strings using more complicated distance measures.

Syntax

COMPARE(*string-1*, *string-2*<,*modifiers*>**)**

Example 5

```
select string1, string3,
       compare(string1, string3) as COMPARE1,
       compare(string1, string3, 'il') as COMPARE2
from test ;
```

results in

string1	string3	COMPARE1	COMPARE2
abc	abc	0	0
ab	a	2	2
bc	BC	-1	0
abc	def	-1	-1
ab	ba	-1	-1

Notice that the first set compare exactly, so the values of COMPARE1 and COMPARE2 are zero. The comparison of the second set indicates that the second character is the first position there is a difference. For the third set, the first compare indicates a difference in the first position and the negative indicates that the first string precedes the second in a sort sequence, but the second compare matches since the modifiers ignore case and trim the leading blanks. Finally, the last two sets do not compare and the first string precedes the second in a sort sequence.

Scan

There is a new scan function, SCANQ, which returns the n^{th} word from a character expression, but ignores delimiters that are enclosed in quotation marks. The default set of delimiters for SCANQ are white space characters, such as blank, tab, and carriage return, which is different from the default list for SCAN. Also, SCANQ does not generate a note in the log if the value of the word count is zero, as SCAN does. Additionally, there are new routines CALL SCAN and CALL SCANQ, which correspond to SCAN and SCANQ but return the position and length of the word.

Syntax

SCANQ(*string*, *n* <,*delimiter(s)*>**)**
CALL SCANQ(*string*, *n*, *position*, *length* <,*delimiters*>**);**

Example 6

```
select string,
       scan(string,2) as SCAN,
       scanq(string,2) as SCANQ,
       scan(string,0) as SCAN0,
       scanq(string,0) as SCANQ0
from test ;
```

results in the following output

string	SCAN	SCANQ	SCAN0	SCANQ0
abc				
123				
abc123				
ABC/123	123			

```
 abc 123   123            123
```

and the following log.

```
699      select string,
700          scan(string,2) as SCAN,
701          scanq(string,2) as SCANQ,
702          scan(string,0) as SCAN0,
703          scanq(string,0) as SCANQ0
704      from test ;
NOTE: Invalid argument 2 to function SCAN. Missing values may be
      generated.
```

SCAN uses '/' as one of its default delimiter, while SCANQ does not. Also, SCANQ does not generate the note in the log when the word count is zero.

Length

SAS®9 has three new length functions. LENGTHC returns the length of a character string, including trailing blanks; LENGTHM returns the amount of memory in bytes that is allocated to a character string; and LENGTHN returns the length of a non-blank character string, excluding trailing blanks, and returns 0 for a blank character string. This last characteristic distinguishes LENGTHN from LENGTH.

Syntax

LENGTHC(*string***)**
LENGTHM(*string***)**
LENGTHN(*string***)**

Example 7

```
select string4,
       length(string4) as LEN,
       lengthc(string4) as LENC,
       lengthn(string4) as LENN
from test ;
```

results in

string4	LEN	LENC	LENN
abc	3	3	3
bc	3	3	3
ab	2	3	2
	1	3	0
a c	3	3	3

Notice that the results are the same if the string contains leading blanks, but can vary if the string contains trailing blanks. Also, the blank string results in a 1 for LENGTH, but a 0 for LENGTHN. Since the variable string4 has a length of 3, the result of LENGTHC will always be 3.

Count

There are two new functions that count the occurrences of a specific substring or specific characters in a string. COUNT looks for a specific substring, while COUNTC searches for specific characters. An optional argument specifies whether to ignore case, trim trailing blanks, or for COUNTC count those characters that are *not* in the list of specified characters.

Syntax

COUNT(*string,substring<,modifiers>***)**
COUNTC(*string,characters<,modifiers>***)**

Example 8

```
select name,
       count(name,'ne') as COUNT,
       count(name,'ne','i') as COUNTcase,
       countc(name,'aeiou','i') as COUNTC,
       countc(name,'aeiou','ivt') as COUNTCnot,
       countc(compress(name),'aeiou','ivt') as COUNTCnot2
from test ;
```

results in

name	COUNT	COUNTcase	COUNTC	COUNTCnot	COUNTCnot2
BOB JONES	0	1	3	6	5
JaNe dOe	0	1	4	4	3
SAM SPADE	0	0	3	6	5
MEL O'DAY	0	0	3	6	5
sally ride	0	0	3	7	6

Without the 'i' modifier the case must match, so COUNT does not recognize 'NE' or 'Ne' as 'ne'. The first COUNTC counts the vowels which for 'BOB JONES' is 3, and the second COUNTC counts the non-vowels including the blank which is 6. The last COUNTC does not count the blank since the COMPRESS function eliminates the blank between the names, resulting in a consonant count.

Proper Names

A nifty function for proper names, PROPCASE, has also been added in SAS®9. This function capitalizes the first letter of each word in a character expression. This works for most names, except for those names that have multiple capitalizations such as those that begin with Mc.

Syntax

PROPCASE(*argument <,delimiter(s)>***)**

Example 9

```
select name, propcase(name) as Proper
from test ;
```

results in

name	Proper
BOB JONES	Bob Jones
JaNe dOe	Jane Doe
SAM SPADE	Sam Spade
MEL O'DAY	Mel O'day
sally ride	Sally Ride

This works well, except for Mel O'day. Specifying " '" as the delimiters would accommodate this case, but not "Mc" or "Mac" names. Additional logic would need to be included to handle these special cases.

Descriptive Statistics Functions

SAS®9 has also added a few descriptive statistics functions. The extensions are not nearly as great as the character functions, but they are still useful.

Central Tendency and Variation

There are three new functions for central tendency, MEDIAN for the median, GEOMEAN for the geometric mean, and HARMEAN for the harmonic mean. Additionally, there are also three new functions for spread or variation, IQR for the interquartile range, MAD for median absolute deviation from the median, and RMS for root mean square. These functions all take a list of variables or values as their arguments.

Syntax

MEDIAN(*value1*<, *value2, ...*>**)**
GEOMEAN(*argument*<,*argument*,...>**)**
HARMEAN(*argument*<,*argument*,...>**)**
IQR(*value-1* <, *value-2...*>**)**
MAD(*value-1* <, *value-2...*>**)**
RMS(*argument*<,*argument*,...>**)**

Example 10

```
select x1, x2, x3, x4, x5,
       median(x1,x2,x3,x4,x5) as Median,
       geomean(x1,x2,x3,x4,x5) as Geometric format=4.1,
       harmean(x1,x2,x3,x4,x5) as Harmonic format=4.1
from test2 ;
select x1, x2, x3, x4, x5,
       iqr(x1,x2,x3,x4,x5) as IQR,
       mad(x1,x2,x3,x4,x5) as MAD format=5.2,
       rms(x1,x2,x3,x4,x5) as RMS format=5.2
from test2 ;
```

results in

x1	x2	x3	x4	x5	Median	Geometric	Harmonic
1	2	4	5	6	4	3.0	2.4
2	5	6	7	2	5	3.8	3.3
4	8	3	2	4	4	3.8	3.4

and

x1	x2	x3	x4	x5	IQR	MAD	RMS
1	2	4	5	6	3	2.00	4.05
2	5	6	7	2	4	2.00	4.86
4	8	3	2	4	1	1.00	4.67

You can see that the median of the first observation is 4. You will have to consult your favorite statistics book or the SAS documentation for the formulas for geometric and harmonic means. The first quartile for the first observation is 2, and the third quartile is 5, so the interquartile range is 5 -2 = 3. The absolute deviations from the median are 3, 2, 0, 1, and 2, so the MAD is 2. The mean of the squares for the first observation is 16.4, so the RMS is 4.05.

Percentiles and Extreme Values

Another set of descriptive statistics functions works with percentiles and extreme values. The PCTL function returns the specified percentile from a list of numeric expressions. You can also specify the percentile definition as in PROC UNIVARIATE by adding a suffix to the function name, such as PCTL5 (which is the default). Two extreme value functions, SMALLEST and LARGEST, return the kth smallest or largest value in a list of numeric expressions. Another function, ORDINAL, although not new in SAS®9 but fits into this theme, returns the kth ordered value.

Syntax

SMALLEST (*k*, *value-1*<, *value-2 ...*>**)**
LARGEST (*k*, *value-1*<, *value-2 ...*>**)**
PCTL<*n*>(*percentage*, *value1*<, *value2, ...*>**)**
ORDINAL(*count,argument,argument*,...**)**

Example 11

```
select x1, x2, x3, x4, x5,
       smallest(2,x1,x2,x3,x4,x5) as SMALL2nd,
       largest(2,x1,x2,x3,x4,x5) as LARGE2nd,
       pctl(25,x1,x2,x3,x4,x5) as P25 format=5.1,
       ordinal(2,x1,x2,x3,x4,x5) as ORDER2,
       ordinal(4,x1,x2,x3,x4,x5) as ORDER4
from test2 ;
```

results in

x1	x2	x3	x4	x5	SMALL2nd	LARGE2nd	P25	ORDER2	ORDER4
1	2	4	5	6	2	5	2.0	2	5
2	5	6	7	2	2	6	2.0	2	6
4	8	3	2	4	3	4	3.0	3	4

The second smallest value for the first observation is 2, the second largest is 5, and the 25th percentile is 2. The second ordinal value is 2 and the fourth ordinal value is 5, which also correspond to the second smallest and second largest values.

Other Functions

There are other new functions that are of special use. These include functions used with the macro facility, and date, truncation, and math functions.

Date

There is a new date function, WEEK, that returns the week value from a date or datetime value. There is a second argument to specify the definition of the first week of the year.

Syntax

WEEK(<*sas_date*>, <*descriptor*>**)**

Example 12

```
select date,
       week(date) as WEEK,
       week(date,'W') as WEEKW,
       week(date,'V') as WEEKV,
       week(date,'U') as WEEKU
from test3 ;
```

results in

date	WEEK	WEEKW	WEEKV	WEEKU
01JAN2005	0	0	53	0
17JUL1972	29	29	29	29
15DEC1993	50	50	50	50
31DEC1995	53	52	52	53
06JUN1944	23	23	23	23

For 01Jan2005, it may be considered in the first week of the 2005 or the last week of 2004, depending on the definition.

Truncation

SAS®9 provides several new truncation functions and a modification to the previous ROUND function. Rounding on a computer can be a bit tricky due to the precision of the decimal values, and the difference between decimal (what we do in our head or on paper) and binary (what the computer does) arithmetic must be considered. The ROUND function, as well as ROUNDE, INT, CEIL and FLOOR, tries to mimic decimal arithmetic, so it makes some adjustments. SAS now provides other functions, ROUNDZ, INTZ, CEILZ and FLOORZ, which do not make these adjustments. ROUND now defaults to a rounding unit of one, if one is not provided. When a value is half-way between rounded values, ROUNDE rounds to the even value, so could be up or down; ROUND always rounds up in this case.

Syntax

ROUNDE (*argument* <,*rounding-unit*>)
ROUNDZ (*argument* <,*rounding-unit*>)

Example 13

```
select x,
       round(x) as ROUND,
       round(x,.1) as ROUND1,
       rounde(x,.1) as ROUNDE
from test3 ;
```

results in

x	ROUND	ROUND1	ROUNDE
3.23	3	3.2	3.2
5.46	5	5.5	5.5
0.5	1	0.5	0.5
4.25	4	4.3	4.2
5.35	5	5.4	5.4

Notice the difference in the fourth observation where ROUND rounds to 4.3 and ROUNDE rounds to 4.2 (since 2 is even).

Coalesce

There are two new coalesce functions, COALESCE and COALESCEC, which return the first non-missing value from a list of arguments. The first is a numeric version, while the second requires character arguments.

Syntax

COALESCE(*argument-1<...*, *argument-n>*)
COALESCEC(*argument-1<...*, *argument-n>*)

Example 14

```
select x, y,
       coalesce(x/y,0) as coal,
       varname,
       coalescec(varname,'N/A') as char_coal length=3
from test3 ;
```

results in

x	y	coal	varname	char_coal
3.23	5.4333	0.594482	b23	b23
5.46	6.532	0.835885	23b	23b
0.5	.	0	_bc	_bc
4.25	0.5	8.5	bc%	bc%
5.35	1.234	4.335494		N/A

Notice that the value of COAL is 0 for the third observation, since x/y is missing. Similarly, CHAR_COAL is ‘N/A’, since varname is missing for the last observation.

Zip Codes

A new function, ZIPCITY, returns the name of a city, given a zip code value. You must have the data set SASHELP.ZIPCODE available for this function to work.

Syntax

ZIPCITY(*zip-code*)

Example 15

```
select zip,
       zipcity(zip) as city length=20
from test3 ;
```

results in

```
zip    city
_____________________________

78746  Austin, TX
10020  New York, NY
50020  Anita, IA
44230  Doylestown, OH
99110  Clayton, WA
```

Cool, huh!

Valid Variable Names

The function NVALID checks the validity of a character string to be used as a variable name. The function returns a value of 1 if the string is a valid variable name, and a 0, otherwise.

Syntax

NVALID(*string*<,*validvarname*>)

Example 16

```
select varname,
       nvalid(varname) as valid
from test3 ;
```

results in

```
varname      valid
_____________________

b23              1
23b              0
_bc              1
bc%              0
                 0
```

In this example, b23 and _bc are valid variable names, but the other values begin with a numeric value, contain a special character other than underscore, or are blank.

Choose

Two new functions, CHOOSEC and CHOOSEN, select a specified item from a list of arguments. The first argument is the item number to be selected, and the rest are items to be selected. CHOOSEC returns character value, while CHOOSEN returns a numeric value.

Syntax

CHOOSEC (*index-expression*, *selection-1* <,...*selection-n*>)
CHOOSEN (*index-expression*, *selection-1* <,...*selection-n*>)

Example 17

```
select x, y, int(uniform(0)*2)+1 as select,
       choosen(calculated select, x,y) as selection
from test3 ;
```

results in

```
        x         y    select  selection
_____________________________________________

     3.23    5.4333         1       3.23
     5.46     6.532         1       5.46
      0.5         .         1        0.5
     4.25       0.5         2        0.5
     5.35     1.234         2      1.234
```

This example randomly chooses from the variables x and y.

Macro

The CALL SYMPUTX assigns a macro variable removing all leading and trailing blanks. This new routine differs from the CALL SYMPUT routine only in its handling of blanks. There are also three functions, SYMEXIST, SYMGLOBL and SYMLOCAL, which check the existence of a macro variable, or whether the macro variable is global or local.

Syntax

CALL SYMPUTX(*macro-variable*, *value* <,*symbol-table*>);
SYMEXIST (*argument*)
SYMGLOBL (*argument*)
SYMLOCAL (*argument*)

Example 18

```
data _null_ ;
       set test(obs=1) ;
       call symput('x1',name) ;
       call symputx('x2',name) ;
run ;
%put &x1:&x2: ;

data _null_ ;
       x=symexist('x1') ;
       y=symexist('y1') ;
       globalx=symglobl('x1') ;
       localx=symlocal('x1') ;
       put x= y= ;
       put globalx= localx= ;
run ;
```

produces in the log

```
100
101  data _null_ ;
102      set test(obs=1) ;
103      call symput('x1',name) ;
104      call symputx('x2',name) ;
105  run ;

NOTE: There were 1 observations read from the data set WORK.TEST.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


106  %put &x1:&x2: ;
BOB JONES   :BOB JONES:
```

and

```
107
108  data _null_ ;
109      x=symexist('x1') ;
110      y=symexist('y1') ;
111      globalx=symglobl('x1') ;
112      localx=symlocal('x1') ;
113      put x= y= ;
114      put globalx= localx= ;
115  run ;
```

```
x=1 y=0
globalx=1 localx=0
```

CALL SYMPUT assigns a macro variable value containing trailing blanks, while CALL SYMPUTX does not. In the second data step, the macro variable x1 exists (SYMEXIST function returns a 1) but y1 does not. Also, x1 is a global, and not local, macro variable.

Create Directories

SAS®9 allows the creation of directories with the DCREATE function. Simply specify the subdirectory and the parent directory.

Syntax

new-directory=***DCREATE***(directory-name<,parent-directory>)

Example 19

```
data _null_ ;
        x=dcreate('scsug05','c:\temp') ;
        put x= ;
run ;
```

results in the log

```
157  data _null_ ;
158       x=dcreate('scsug05','c:\temp') ;
159       put x= ;
160  run ;

x=c:\temp\scsug05
```

This creates the subdirectory scsug05 under the parent directory c:\temp.

Logic Functions

Two new functions, IFC and IFN, return either character or numerical expressions based on a conditional expression. These could be used in place of IF/ELSE logic and could be used in a WHERE statement.

Syntax

IFC(*logical-expression*, *value-returned-when-true*, *value-returned-when-false*
<,*value-returned-when-missing*>)
IFN(*logical-expression*, *value-returned-when-true*, *value-returned-when-false*
<,*value-returned-when-missing*>)

Example 20

```
data check ;
        set test3 ;
        st_size= ifc(zipstate(zip) in ('TX','NY','CA'),'Big','Little') ;
        city= zipcity(zip) ;
run ;
```

A listing of the data

Obs	city	zip	st_size
1	Austin, TX	78746	Big
2	New York, NY	10020	Big
3	Anita, IA	50020	Little
4	Doylestown, OH	44230	Little
5	Clayton, WA	99110	Little

The first two cities are in Big states, while the last three are in Little states.

Formats

Two new functions, VVALUE and VVALUEX, return the formatted value of a variable or a character expression. These functions have similar functionality to the PUT statement, but apply the current format.

Syntax

VVALUE(*var*)
VVALUEX(*expression*)

Example 21

```
data check ;
        set test3 ;
        length newdate newdate2 $ 9 ;
        newdate=vvalue(date) ;
        newdate2=vvaluex('da'||'te') ;
run ;
```

A listing of this data is

Obs	date	newdate	newdate2
1	01/01/2005	01JAN2005	01JAN2005
2	07/17/1972	17JUL1972	17JUL1972
3	12/15/1993	15DEC1993	15DEC1993
4	12/31/1995	31DEC1995	31DEC1995
5	06/06/1944	06JUN1944	06JUN1944

The DATE variable had a DATE9. format, and this is the format VVALUE and VVALUEX used for the new variables. Note that VVALUE requires a variable name, while VVALUEX can take an expression.

Setting values to missing

There is a new convenient way to set a list of variables to missing through the CALL MISSING routine. This can be used in place of a DO loop setting the variables to missing.

Syntax

CALL MISSING(*varname1*<, *varname2*, ...>);

Example 22

```
data check2 ;
        set test3 ;
        if _n_=1 then
                call missing(n, sumx, sumy) ;
        if n>2 then call missing(n, sumx, sumy) ;
        n + 1 ;
        sumx + x ;
        sumy + y ;
run ;
```

A listing of the data is

Obs	n	x	sumx	y	sumy
1	1	3.23	3.23	5.4333	5.4333
2	2	5.46	8.69	6.5320	11.9653
3	3	0.50	9.19	.	11.9653
4	1	4.25	4.25	0.5000	0.5000
5	2	5.35	9.60	1.2340	1.7340

The variables n, sumx, and sumy are set to missing the first time through the DATA step, and then again once n is greater than 2.

Conclusion

This paper was not intended to be either an exhaustive list or to provide a full explanation of the new functions in SAS®9. However, it is hoped that seeing the usefulness of some of these functions will spur you to check them out in the SAS documentation. Full documentation of these functions and routines are available in the Online Doc for SAS®9. Also, most of these functions and routines are described in more detail in SAS Functions by Example, by Ron Cody.

References

Cody, Ron, **SAS Functions by Example**, 2004, SAS Institute, Inc., Cary, NC.

Contact information:

Keith Cranford
keith.cranford@cs.oag.state.tx.us