Keeping Up With the FUN: New Functions in SAS 9

Deb Cassidy, Dublin, OH

ABSTRACT

This presentation will cover some of the many new data step functions found in SAS 9. Some of these functions will add new functionality. Other functions will be faster - either in the processing time or less typing. Some new functions are very similar to functions found in other languages.

(Note: In the actual presentation, this was not presented in the traditional lecture format. It was done in a FUN format to fit the title.)

INTRODUCTION

The first big question – when is SAS 9 available? You can get 9.0 now by request. SAS 9.1 will be available to select customers in 3rd quarter 2002. Everyone should have it by 1st quarter 2004. As of the latest information at the time this paper was written, it was still being decided if there would be a mass shipment or some other method in place to obtain SAS 9.1.

Are you wondering why you should care? Version 6 HELP lists 19 categories with 236 functions. SAS 9 HELP lists 26 categories with 452 functions – quite a jump. If you are used to looking up functions by category, you will want to know that category names have been modified and some functions were reclassified.

Why are the new functions? Some offer new functionality. Others have faster system performance or a faster way for you to write and review the code. Still other functions were added to be similar to other languages.

This paper will cover some of the many functions. The best place to find out about all the functions is SAS 9 Help.
Note: □ is used to represent a blank in the code samples when it may not be obvious.

PUTTING THINGS TOGETHER

You have a first name and last name and now you want to create a single field in the form “Doe, Jane”. The old way was

```
OLDWAY=LEFT(TRIM(LAST)) || “, □” ||
LEFT(TRIM(FIRST));
```

For some people, the hardest part was figuring out how to specify the concatenate symbol because there is not such a symbol on the keyboard!

Here are four new concatenate functions. But which one really gives you what you need?

```
CAT1=CAT(LAST,”,□”,FIRST);
CAT2=CATT(LAST,”,□”,FIRST);
CAT3=CATS(LAST,”,□”,FIRST);
CAT4=CATX(“,□“,LAST,FIRST);
```

At first glance, you might think they all would work. The first three statements say to concatenate everything together. However, CAT does not work in this case because you have trailing blanks in your original fields that are kept. CATT trims trailing blanks. CATS is similar but strips off both leading and trailing blanks. Too bad they both also trim the blank after the comma that you wanted to keep. CATX to the rescue. In this function, the delimiter is specified first. Leading and trailing blanks are removed for the other fields but not from the delimiter. You will need to remember than the concatenate functions are like many of the functions in previous versions - the default length is $200 and should normally be specified.

SELECTING SINGLE CHARACTERS
The old method was to use

OLDWAY=**SUBSTR**(FIELD,5,1);

Now you can use either of the following:

LETTER=**CHAR**(FIELD,5);
FRSTLTR=**FIRST**(FIELD);

Some people will say there is not much difference. These are functions that were probably added for efficiency or to be consistent with other languages.

BLANKS
Do you have blanks in your fields that you want to remove? The old way was

OLDWAY=**LEFT**(**TRIM**(FIELD));

While that was not complicated, the following makes it even easier.

NEW=**STRIP**(FIELD);

Some people might want to put those blanks back in as in the following example.

ADDBLANKS=**SUBPAD**(FIELD);

THE DREADED DIVISION BY 0
You code says

Y=A/B;

How could something so simple cause problems? Easy, if you did not now you had records with a value of 0 for B.

You get a message telling you were the divison by zero occurred.

```
NOTE: Division by zero detected at line 79
    column 7.
```
In most cases, it shows up way too many times in the log.

The old way to get rid of these messages was to use the following logic:

```
      if B NE 0 then Y=A/B;
else Y=.;
```

The much simpler way is

Y=**DIVIDE**(A,B);

The bad news is that DIVIDE is not available until SAS 9.1 so you may still be waiting a few months for an easy way to get rid of those nasty messages.

FINDING A STRING
Do you need to find a string within a field? The old code was

OLD=**INDEX**(FIELD,"HI");

This returned the column of the first letter of the first occurrence of the string. The new function works in much the same way and corresponds to functions in other languages.

NEW=**FIND**(FIELD,"HI");

The FIND function also has some other options. You can specify "I" to ignore case or "T" to trim leading and trailing blanks. Here are a few examples.

A='abcedfghijhih';
B= ' xy ';

Statement	Returns
Old1=index(a,'hi');	8
Old2=index(a,'HI');	0
Old3=index(upcase(a),'HI');	8
New1=find(a,'hi');	8
New2=find(a,'HI');	0
New3=find(a,'HI','i');	8
New4=find(b, ' xy ','t');	2

The other parameter you can specify for this function is the starting point and the direction. However, you need to make sure you understand what happens when you go in a negative direction. Here are a few examples to show that negative simply means find the first occurrence starting from the right but still "reading from the left". It does not mean the field is checked as though it had been reversed.

Statement	Returns
New5=find(a,'hi',**11**);	11
New6=find(a,'hi',**-11**);	8
New7=find(a,'hi',-13);	11
New8=find(a,'hi',-15);	11

SPOTTING TYPOS
Do you need to check your data to make sure the values are legitimate. For example, you need to check that only the letters A-Z appear in a field.

OLDWAY=**VERIFY**(FIELD,
'ABCDEFGHJKLMNOPQRSTUVWXYZ')
;

While it looks simple, you had to type every letter from A-Z so it might be easy to miss one! (Did you notice if a letter was missing in the above statment?)

The new method is much easier:

NEW=**NOTALPHA**(FIELD);

This will return the position of the first character that is not alpha. You can also easily find the first character that is alpha by using:

NEW=**ANYALPHA**(FIELD);

Are you more intersested in numerics than alpha? How about punctuation characters? The "not/any" functions are listed here along with some examples.

ALNUM - alphanumeric
CNTRL - control character
DIGIT - digit
FIRST – first character of a variable name
GRAPH – graphical character
LOWER – lower case
NAME – character in a variable name
PRINT – printable character
PUNCT – punctuation character
SPACE – white-space character
UPPER – upper case letter
XDIGIT – hex digit character

A= '$_ab12A: □5';

Statement	Result
A1=anyalnum(a);	3
A2=anycntrl(a);	0
A3=anydigit(a);	5
A4=anyfirst(a);	2
A5=anygraph(a);	1
A6=anylower(a);	3
A7=anyname(a);	2
A8=anyprint(a);	1
A9=anypunct(a);	1
A10=anyspace(a);	9
A11=anyupper(a);	7
A12=anyxdigit(a);	3

A= '$_ab12A: □5';

Statement	Result
X1=notalnum(a);	1
X2=notcntrl(a);	1
X3=notdigit(a);	1
X4=notfirst(a);	1
X5=notgraph(a);	9
X6=notlower(a);	1
X7=notname(a);	1
X8=notprint(a);	0
X9=notpunct(a);	3
X10=notspace(a);	1
X11=notupper(a);	1
X12=notxdigit(a);	1

HOW LONG IS IT?
Are you one of the many folks who just didn't like the fact that

LEN=**LENGTH**("");

returned 1? If there isn't anything there, how could it possible have a length of one? If you have

LEN=**LENGTH**("A");

you also get a length of 1.

Well now, you have some choices. **LENGTH** returns the length excluding trailing blanks but returns a 1 if there isn't anything in the string.

LENGTHN returns the length excluding trailing blanks but unlike LENGTH, it will return a 0 for blank character string.

LENGTHC returns the length including trailing blanks. It can never return 0.

LENGTHM returns the amount of memory that is allocated for the string.

A="";
B="LONGER";
C="LONG WITH BLANKS   ";
Length D $200;
D="EXTRA LONG";

STATEMENT	RETURNS
A0=LENGTH(A);	1
A1=LENGTHN(A);	0
A2=LENGTHC(A);	1
A3=LENGTHM(A);	1
B="LONG";	
B0=LENGTH(B);	4
B1=LENGTHN(B);	4
B2=LENGTHC(B);	4
B3=LENGTHM(B);	4
C0=LENGTH(C);	16
C1=LENGTHN(C);	16
C2=LENGTHC(C);	24
C3=LENGTHM(C);	24
D0=LENGTH(D);	10
D1=LENGTHN(D);	10
D2=LENGTHC(D);	200
D3=LENGTHM(D);	200

SAME OR NOT?
Do you need to check if two fields are the same? Then **COMPARE** is for you (do not confuse it with the PROC which works on datasets).

X='name';
Y='naem';

COMPARE will return the location of the first character that is different.

Z=**COMPARE**(X,Y);

In the above example, it is pretty obvious that the 3rd letter differs so COMPARE returns 3.

However, if you switch the order of your fields to

Z=**COMPARE**(Y,X);

then COMPARE returns -3. Why?
The sign of the result tells you which field comes first based on your sort sequence. It is negative if the first field listed would sort before the second field listed. Otherwise, it is positive. In this case, on a Windows system, "naem" would sort before "name" so the result was negative. Remember, sort order can vary by operating system and you can overrice the sort order with options.

You have some other options. For example, you can ignore case in doing your comparison, remove leading blanks or quotation marks, and truncate fields to the shorter length.

Here are a few more examples.

X	Y	Comparison	Result
Name	naem	(x,y)	-1
Name	naem	(y,x)	1
Name	naem	(x,y,'i')	3
X	X	(x,y)	-1
X	X	(x,y,'l')	0
'ab'	'AB'	(x,y)	2
'ab'	'AB'	(x,y,'n')	0
ABC	AB	(x,y)	3
ABC	AB	(x,y,':')	0

SMALLEST & LARGEST
A variation of this presentation was given at SUGI except the audience was only asked a couple questions including one about the new **LARGEST** function. Most people guessed the wrong answer and you could hear the collective moan when they were told the right answer. That was the motivation behind doing this paper in a "fun" format and emphasizing the moral of the story – assuming how a function works without looking into the details can be hazardous to your results!

The audience was asked for the results of the function

LARGE2=**LARGEST**(2,A,B,C,D);

when the following values were used.

A=1
B=3
C=5
D=7

Some in the SUGI audience replied 7. Others realized the 2 indicated the "second item" and replied 3. Others replied "D" or "B" which is the variable name. The answer is really 5. Why? Those who thought the 2 indicated the "second item" were right except it isn't the second item in order but the second largest item. In this case, 7 is the largest value and 5 is the second largest.

The related **SMALLEST** function works in the same way.

UP OR DOWN
There have always been various rules for rounding and they vary by region as well as industry. I was always taught that if the value ends in 0-4, you round down. Otherwise, you round up for 5-9. However, other people were taught to that values ending in 5 are rounded up or down depending on whether the previous digit is odd or even.

The traditional **ROUND** rounds the first argument to the nearest multiple of the second argument, or to the nearest integer when the second argument is omitted. The new **ROUNDE** returns and even multiple when the first argument is halfway between the two nearest multiples. **ROUNDZ** is like **ROUND** but it does not use fuzzing.

	ROUND	ROUNDE	ROUNDZ
(123.45,.1)	123.5	123.4	123.4
(123.55,.1)	123.6	123.6	123.5
(3.156,.003)	3.156	3.156	3.156
(3.157,.003)	3.156	3.156	3.156

STILL A LITTLE FUZZY?
The fuzz factor says that if things are close, SAS will consider them equal. There are several procs which allow you to specify the fuzz factor. Normally, the fuzz factor is 1E-12. Many functions used the fuzz factor by default. There are now functions which use 0 as a fuzz factor. These include **FLOORZ**, **CEILZ**, **INTZ** and **MODZ**. However, there is a caution in the HELP stating that you may be unexpected results!

Here are a few examples for the value 3.55943

	Regular	Z
CEIL	4	4
FLOOR	3	3
INT	3	3
MOD,2	1.55943	1.55943

Are you wondering why they are the same? I kept trying to come up with an example and then I finally read the HELP a little closer.

Let's say your value is in exponential notation such as (1.-1.e-13)

	Regular	Z
CEIL	1	1
FLOOR	1	0
INT	1	0
MOD,2	1	1

This time you really do get different results using the fuzz factor and not using it.

NO MORE PUT?
Do you ever need to convert a numeric value to a character? The old way was:

```
OLDWAY=PUT(Y,10.2);
```

where Y is a numeric value and OLDWAY becomes character with the length dependent upon the length you specify in the format. In this case, the length of the character field is 10.

The new way uses the **VVALUE** function

```
FORMAT Y 10.2;
NEWWAY=VVALUE(Y);
```

Well, this takes more code to write so the question is whether it is more efficient. I'll leave that up to you to determine. I will tell you there is a negative side to this method. The default length for the new variable is $200!

However, there is a related function which could prove to be quite useful. Here is an example:

```
DATA MYVAL;
FIELD1=17.5;
FIELD2=21.2;
FORMAT FIELD1 5.2 FIELD2 Z6.2;

MYFIELD='FIELD1';
MYVALUE=VVALUEX(MYFIELD);
```

When MYFIELD is set to FIELD1, the result in MYVALUE is 17.50. When MYFIELD is set to FIELD2, then the result in 021.20. The value of MYVALUE will be determined by MYFIELD. I see great use when something in your code or data specifies what MYFIELD should be for each record.

If you aren't familiar with the Z. format, it keeps the leading zeroes. At first glance, I expected 2 leading zeroes since I specified a length of 6 and 21.20 only accounts for 4 digits. Then I remembered the decimal is counted as part of the length. One other point – this is another function which defaults to length $200 so don't forget to specify a length statement.

MORE!

This is only the tip of the iceberg as far as new functions. Some are more useful in some industries that others. The examples I picked are all related to papers I have written in the past - and now need to update to reflect SAS 9!

HOW TO LEARN MORE

The Online Documentation contains lots of help on these new functions. Depending upon your site, you may be able to access it locally. If not or you just can not wait until they install SAS 9 at your site, then check out **SUPPORT.SAS.COM**. You need to register but access is free.

Need some customized help in figuring out a function and what it is doing? Then try **SAS-L**, an electronic user group that covers the entire world! There are several ways to get access to SAS-L. There are numerous SUGI and SESUG papers which explain the options. Check out Joe Kelley's paper at this SESUG to learn how SAS-L functions behind the scenes. Joe is not only an active SESUG member but the "man behind the screen" for the University of Georgia which is one of the host sites for SAS-L. My preferred method of reading SAS-L is to use

http://www.listserv.uga.edu/archives/sas-l.html

but some people prefer newsgroups or getting e-mail.

CONCLUSION

There are lots of new functions in SAS 9. Unless you want to be considered out-of-date, you'll want to review the functions and see which ones can improve your work. As I pointed out in some of the examples, though, make sure you understand the function, what it does and how to use the parameters correctly or you could end up with wrong results.

CONTACT INFORMATION

Deb Cassidy
Deb.cassidy@cardinal.com
614-757-7136