Paper 81-25

My Favorite Functions

Pete Lund, WA State Caseload Forecast Council, Olympia, WA

ABSTRACT

Just like Uncle Martin had to sometimes raise his antennae and wiggle his finger to get a job done, sometimes it is helpful to enhance your SAS toolkit with functions and utilities you've written yourself. This paper briefly discusses a number of functions I've written that have become invaluable tools.

SAS is a powerful, robust language. There are functions and formats that allow you to read, write and manipulate data almost anyway you need to. However, every once in a while you come up with a situation that you need a few lines of code or a few embedded functions to handle. This is where the SAS macro facility can come to the rescue.

DATE FUNCTIONS

SAS has numerous date functions, formats, and informats. There are times, however, when you can't quite get what you want. For example, the INTNX function will increment a SAS date value by a specified number of intervals. However, the result is always the first (or last or middle) day of the interval. For example, if x=4/1/1999, the following

```
Y = intnx('year',x,3)
```

will result in 1/1/2002 – the first day of the year that 3 years from the base date. What if we need a function which will return the same date, x number of years into the future. Here's a little macro function to do just that.

```
%macro AddYears(base,add);
  intnx('month',&base,&add*12) + (day(&base)-1)
%mend;
```

The %AddYears macro increments the date to the right month by incrementing by months (years * 12) instead of years. Now that we're at the first day of the right month, just add the day value of the date passed (-1, since we're already on the first).

Another common date-related task is to determine the fiscal year of a date variable. After years of creating little formats containing date ranges or parsing out the components of the date and determining the fiscal year, this little function dawned on me.

```
%macro fy(date,start=7);
  year(&date) + (month(&date) ge &start and &start ne 1)
%mend;
```

First, get the year of the date. Then, add one to the value if the month of the date is greater than our fiscal year start month (the default is 7, July). The boolean part of the expression, month(&date) ge &start, will return 1 if true and 0 if false. So, if the date is Oct 10, 1999, the year is 1999 and 10 > 7, so add 1, gives a fiscal year value of 2000.

The start parameter allows one to use any month as the base for the fiscal year. If you pass a 1 for the start month (same as calendar year), no adjustment is made to the year value.

STRING FUNCTIONS

The SAS language provides a SUBSTR function for snipping a piece of a character variable. It's syntax is similar to the MID function in many other programming languages, specifying the variable name, start position and length of the sub-string. Languages which use a MID function also have a LEFT and a RIGHT function, to extract the leftmost and rightmost characters, respectively.

A RIGHT function, however, is often very handy and will save a bit of coding. Say, for instance, that you have a variable that can be variable length and contains a status code as the last 2 characters. The following function can be called to extract those characters:

```
%macro right(vn,len);
  reverse(substr(reverse(&vn),1,&len))
%mend;
```

Assume CatID has a value of “A123AC.” The following

```
status = %right(CatID,2);
```

would yield, “AC” as a result. Again, we’ve embedded three functions.

1. reverse(“A123AC”) = “CA321A”
2. substr(“CA321A”,2) = “CA”
3. reverse(“CA”) = “AC”

Until now, all the functions we’ve looked at contained or created a single SAS code segment. This allowed the macros to be treated as functions and called in a line of SAS code. There are often times when function-like capabilities require more than one line of SAS code. These statements can still be wrapped in a macro and thought of as a function. However, the call to the macro will be a stand-alone statement rather than part of an `IF` or `WHERE` statement.

An example of this type of function is found below. It is a macro that accepts a character variable as an argument and places a mixed case version of that variable in a return variable. For example, JOHN Q PUBLIC would be returned as John Q Public.

```
%macro mixed(InString,OutStr);
  drop _i _NumWrds _Word _NewWord _OutStr;

_NumWrds = (length(&InString) -
length(compress(&InString,' '))) + 1;

 _OutStr = &InString;
  _OutStr = '';

do _i = 1 to _NumWrds;
   _Word = scan(&InString,_i,' ');
   if length(_Word) gt 1 then
     _NewWord =
upcase(substr(_Word,1,1))||lowcase(substr(_Word,2));
   else _NewWord = upcase(_Word);
   _OutStr = trim(left(_OutStr))||' '||_NewWord;
 end;

 &OutStr = _OutStr;
%mend;
```

The macro loops through each word of the passed value, using the SCAN function, and, using the UPCASE and LOWCASE functions, sets the case of each word in the string. So, for the example above, if TheName = “JOHN Q PUBLIC”:

```
%Mixed(TheName,MixName);
```

MixName would now equal “John Q Public”.

MISCELLANEOUS FUNCTIONS

The first “function” I wrote in SAS was a substitute for the IN operator, which is often referred to as a function. I had a need to be able to check a character variable against a list of values of varying lengths. In addition, at times only the first few characters of the string needed to match.

```
%macro in() / parmbuff;
 %let parms =
%qsubstr(&syspbuff,2,%length(&syspbuff)-2);

 %let var = %scan(&parms,1,%str(,));

 %let numparms = %eval(%length(&parms) -
%length(%sysfunc(compress(&parms,%str(,)))));

 %let infunc = &var eq %scan(&parms,2,%str(,));

 %do i = 3 %to (&numparms + 1);
  %let thisparm = %scan(&parms,&i,%str(,));
  %let infunc = &infunc or &var eq &thisparm;
 %end;

 (&infunc)
%mend;
```

For example, I want to check a variable, CPT, against the following values 4300, 4301, 44xx, 451x. I can’t use the colon operator (:”44”) in the IN operator and I don’t really want to spell out “4401”,’4402”, etc. and I really don’t want to write a bunch of IF…OR…OR…OR logic either.

The little macro above automates the process. Our example would be coded as follows:

If (cpt,’4300’,’4301’,:’44’,:’451’) then…

would actually generate the following code:

if cpt eq ‘4300’ or cpt eq ‘4301’ or cpt eq :’44’ or cpt eq :’451’ then….

Any number of functions can be embedded in the list of values. The macro works by parsing out the variable name and the values to check and generating the **varname eq value** pairs. It is beyond the scope of this paper to discuss the workings of this particular macro. A more detailed discussion of this and other user-written functions can be found in “Need More Functionality: Using the SAS Macro Language to Create User-Written Functions” (Lund, 1998).

I hope this has piqued your interest in writing your own functions and utilities. It's a great way to learn and can be a real time saver as you write more complex application.

TRADEMARKS

REFERENCES

Lund, Peter, "Need More Functionality: Using the SAS Macro Language to Create User-Written Functions", *Proceedings of the Seventeenth Annual Pacific Northwest SAS Users Group Regional Conference*, Portland, OR: 1998.

SAS Institute Inc., *SAS Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute, Inc., 1990.

SAS Institute Inc., *SAS Macro Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute, Inc., 1997.

AUTHOR CONTACT INFORMATION

An electronic version of this paper and all the macros can be obtained, via e-mail, from the author.

Pete Lund
Washington State Forecast Council
515 E 15th Ave SE
Olympia, WA 98504-0962
(360) 902-0086 voice
(360) 902-0084 FAX
peter.lund@cfc.wa.gov