Paper 044-29

The Very Valuable Variable Value Functions

Howard Schreier, U.S. Dept. of Commerce, Washington DC

ABSTRACT

Sometimes it is useful, within a DATA step, to inspect and utilize a variable's formatted value. In earlier versions of SAS® software, this was clumsy. Version 9 introduces the VVALUE function, which makes the process simpler. The related VVALUEX function makes the operation data-driven (rather than code-driven) and thus makes it possible to reference data by name at execution time.

INTRODUCTION

A format is that attribute of a SAS variable which governs how values of that variable are displayed. SAS has a library of built-in formats. You can also make use of the Format Procedure (PROC FORMAT) to add your own.

The VVALUE function takes, as its only argument, the name of a variable or an array reference. It returns a character string containing the formatted value of the variable referenced. The VVALUEX function works similarly, but its single argument is a character expression which must evaluate to the name of a variable.

EXAMPLE PROBLEM

To illustrate the usage of the VVALUE and VVALUEX functions, we will first define a simple categorical numeric format:

```
proc format;
value myfmt (default=15)
            1 = 'Yes'
            2 = 'No'
            3 = 'Don''t Know'
            other = 'N/A'
            ;
run;
```

Next we will build a tiny dataset (one observation with five variables) and associate different formats with each variable:

```
data demo;
retain a b c d e 3 ;
format a 4.1
       b 5.2
       d date9.
       e myfmt12.
       ;
put (_all_)(=) ;
run;
```

Note that one variable (C) is not explicitly assigned a format. This does not mean that it is without one. Rather, the default numeric format (BEST12.) is in effect.

When this DATA step runs, the PUT statement places the following line in the log:

```
a=3.0 b=3.00 c=3 d=04JAN1960 e=Don't Know
```

This demonstrates the functionality of formats; recall that each of these numeric variables has the same value (3).

As we have seen, the PUT statement (or at least this form of the PUT statement) automatically applies the formats which have been associated with the variables. But what if it's necessary to reference these formatted values within the DATA step? For example, one might want to test their content or embed them in longer strings.

The PUT function is applicable in this situation. A great thing about the PUT function is that one can specify any appropriate format. A bad thing about the PUT function is that one must specify a format. Why should that be mandatory when every variable has associated with it (either by earlier declaration or by default) a format?

Here is a concrete example. The task is to write the five variables to the log, one per line, with decimal points aligned. If there is no decimal point, the right alignment should be just clear of where the decimal point would be. In other words, the display should look like this:

```
               3.0
               3.00
               3
        04JAN1960
       Don't Know
```

Here's one way to do this:

```
data _null_;
set demo;
put +11 +1 +1 a 4.1      -r;
put +10 +1 +2 b 5.2      -r;
put  +3       c best12.  -r;
put  +6       d date9.   -r;
put  +3       e myfmt12. -r;
run;
```

We have hard coded the formats which were already associated with the variables. The "+" pointer controls compensate for the varying widths of the formats and for the space taken by decimal points and decimal digits; The "r" modifier specifies right alignment.

VVALUE

Since formats were previously associated with the variables, it should be possible to implicitly reference these and avoid all of the hard coding (format specifications themselves and pointer controls derived from them).

Here is code:

```
data _null_;
set demo;
array abcde (5) a b c d e ;
do i = 1 to 5;
   charvalue = vvalue(abcde(i) );
   indent = 16-length(strip(charvalue) );
   decimalpart = scan(charvalue,2,'.');
   if not missing(decimalpart) then
    indent = indent + length(decimalpart) + 1;
   put @indent charvalue;
   end;
run;
```

The DO loop processes each variable, as follows.

- The VVALUE function is used to load the formatted value of a variable (referenced through an array) into the character variable CHARVALUE.
- The indentation is tentatively set to 16 less the length of the formatted value.
- The substring which is to the right of the decimal point is isolated in DECIMALPART. The SCAN function returns a missing value if there is no decimal point.
- If there is a decimal substring, the indentation is increased to accommodate it.
- The result is written to the log, using the computed indentation.

In earlier versions of SAS (before Version 9), this was a bit more clumsy. It would have been necessary to use the VFORMAT and PUTN functions together to do what VVALUE does here, and also to use the LEFT and TRIM functions together to do what the STRIP function does.

Unfortunately, the code does not work. There is a bug in Version 9 which causes VVALUE to return incorrect results when the argument is an array reference. Version 9.1 is expected to correct this. Meanwhile, here is a workaround which uses the new functions but avoids array references.

```
data _null_;
set demo;
charvalue = vvalue(a) ; link display ;
charvalue = vvalue(b) ; link display ;
charvalue = vvalue(c) ; link display ;
charvalue = vvalue(d) ; link display ;
charvalue = vvalue(e) ; link display ;
return;
display :
indent = 16-length(strip(charvalue) ) ;
decimalpart = scan(charvalue,2,'.');
if not missing(decimalpart) then indent = indent +
length(decimalpart) + 1;
put @indent charvalue;
return;
run;
```

VVALUEX

The VVALUEX function is similar to the VVALUE function in terms of what it does, but it is much different in that it requires as its argument a character expression which evaluates to the name of a variable. For example, if MYVAR is the name of a variable, coding

```
vvaluex("myvar")
```

is effectively the same as coding

```
vvalue(myvar)
```

Of course this is a trivial usage in that the expression is just a constant. The power comes in using formulas to defer until execution time the designation of the variable whose value is to be retrieved and formatted.

Here is code to solve the given problem using the VVALUEX function. We could hard code the progression of variables' names ("A", "B", "C", "D", "E"). Instead, to demonstrate the capacity to make this data-driven, they will be read. Here is the code:

```
data _null_;
if _n_=1 then set demo;
infile cards;
input chosenvar $ ;
charvalue = vvaluex(chosenvar) ;
indent = 16-length(strip(charvalue) );
decimalpart = scan(charvalue,2,'.');
if not missing(decimalpart) then
 indent = indent + length(decimalpart) + 1;
put @indent charvalue;
cards;
c
e
a
d
;
```

The results:

```
                3
      Don't Know
                3.0
       04JAN1960
```

The changed order and the omission of the variable B reflect the data stream read by the INPUT statement. If all five variable names had been specified, in alphabetical order, the results would have been exactly as shown earlier.

This example of VVALUEX usage was designed to parallel the example used for VVALUE, but that actually hides much of the versatility of VVALUEX. What we actually have here is a direct way to reference data items, at execution time, by name.

CONCLUSION

The VVALUE function makes it possible for SAS code to reference the formatted value of a variable without having to specify or retrieve the variable's format. The VVALUEX function goes a step further by allowing the process to be data-driven rather than code-driven.

REFERENCES

SAS Institute Inc., *SAS OnlineDoc® 9*, http://v9doc.sas.com/sasdoc/

CONTACT INFORMATION

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

Howard Schreier
U.S. Dept. of Commerce
Mail Stop H-2015
Washington DC 20230

202-482-4180

Howard_Schreier@ita.doc.gov