Paper 245-2012

Using the New Features in PROC FORMAT

Rick Langston, SAS Institute Inc., Cary, NC

ABSTRACT

This paper describes several examples using functions as labels within PROC FORMAT definitions. Also described is the new feature allowing for Perl regular expressions for informatting data, as well as other new features in PROC FORMAT for SAS® 9.3.

INTRODUCTION

SAS® 9.3 has now been released, and the FORMAT procedure has a number of new features with this release. This paper discusses the new features and shows examples of each. The new features are: Perl regular expressions in informats, locale-sensitive format catalogs, a day count directive in picture formats, the DATETIME_UTIL DATATYPE value for the utility industry, and functions in label specifications.

PERL REGULAR EXPRESSIONS IN INFORMATS

Perl regular expressions have been usable within SAS for several releases, via the PRX* functions of the DATA step, including PRXPARSE and PRXCHANGE. Now the functionality is available to users of informats being defined with the INVALUE statement of PROC FORMAT.

Instead of the informat processing performing an exact match on a text string, the Perl regular expression is applied to the input text, and if there is a match according to the rules of Perl, then the value on the right-hand side of the equal sign will be used as the input value. The (REGEXP) option is given after the quoted Perl regular expression to indicate that it is to be treated this way. For example, consider a Perl regular expression that will verify that input text is of the form of a time value:

```
proc format;
     invalue xxx (default=20)
              '/(\d+):(\d\d)(?:\.(\d+))?/' (REGEXP) = [time8.];
     run;
```

The Perl regular expression tells us to look for one or more digits, followed by a colon, followed by 2 digits, then followed by a colon or a dot, then with one or more digits. If the informat encounters such a string, then the TIME8. informat given in the brackets on the right-hand side of the equal sign will be invoked in order to convert the text string into a SAS time value.

We can add more expressions and more informats, and allow a DATA step to evaluate input text and pass it to the proper informat, whether it be TIME8., DATE7., or DATE9., as seen below:

```
proc format;
     invalue xxx (default=20)
              '/(\d+):(\d\d)(?:\.(\d+))?/' (REGEXP) = [time8.]
             '/\d\d\D\D\D\d\d/' (REGEXP) = [date7.]
             '/\d\d\D\D\D\d\d\d\d/' (REGEXP) = [date9.]
              other=_error_;
     run;

data _null_; input x: xxx. @@; put x= x=date9. x=time10.1; cards;
10:24
01jan02
10
abc
10:24:35
10:35:24.2
10:38
01jan2002
run;
```

OUTPUT 1: Output from the DATA step

```
x=37440 x=04JUL2062 x=10:24:00.0
x=15341 x=01JAN2002 x=4:15:41.0
NOTE: Invalid data for x in line 19 1-2.
x=. x=. x=.
x=. _ERROR_=1 _N_=3
NOTE: Invalid data for x in line 20 1-3.
x=. x=. x=.
20         abc
x=. _ERROR_=1 _N_=4
x=37475 x=08AUG2062 x=10:24:35.0
x=38124.2 x=18MAY2064 x=10:35:24.2
x=38280 x=21OCT2064 x=10:38:00.0
x=15341 x=01JAN2002 x=4:15:41.0
```

With the first input value, 10:24, the Perl regular expression corresponding to TIME8. gets a match, so the TIME8. informat is used and we see that 10:24:00 is the formatted value when TIME8. Is used. 04JUL2062 just happens to be the formatted value when the time value is passed to a DATE format. In the case of 01jan02, the Perl regular expression corresponding to DATE7. gets a match, so the DATE7. informat is used and we see that 01JAN2002 is the formatted value when DATE9. Is used. 4:15:41.0 just happens to be the formatted value when the date value is passed to a TIME format. The third value, 10, does not match with any Perl regular expression, so the OTHER=_ERROR_ clause is used and we get an “Invalid data” message. Likewise for the value abc. The remaining input values match up properly with the TIME or DATE informats.

You can use the substitute feature of Perl regular expressions by using the REGEXPE option. If this option is used, the result of the substitution is passed to the expression on the right-hand side of the equal sign. This works similarly to the PRXCHANGE function.

In this example, the input value of abc will be converted to def, while all other input values will remain as they are.

```
proc format;
     invalue $abc2def (default=20)
             's/abc/def/' (regexpe) = _same_
             other                  = _same_;
     run;
data _null_; input x: $abc2def. @@; put x=; datalines;
abc xyz
;
```

OUTPUT 2: Output from the DATA step

```
x=def
x=xyz
```

Here we expand on this example, and show that you can chain expressions together.

```
proc format;
     invalue $abc2def (default=20)
             's/abc/def/' (regexpe) = [$def2ghi.];
     invalue $def2ghi (default=20)
             's/def/ghi/' (regexpe) = [$ghi2jkl.];
     invalue $ghi2jkl (default=20)
             's/ghi/jkl/' (regexpe) = [$jkl2mno.];
     invalue $jkl2mno (default=20)
             's/jkl/mno/' (regexpe) = _same_;
     run;

data _null_;
     x=input('abc',$abc2def.);
     put x=;
     run;
```

OUTPUT 3: Output from the DATA step

```
x=mno
```

LOCALE-SENSITIVE FORMAT CATALOGS

If you need to create formats that have different versions for different languages or locales, you can save the different versions in format catalogs with locale-specific suffixes. Then you can use locale-independent SAS code to reference the proper catalogs. The new LOCALE option of PROC FORMAT indicates that the format catalog will be named with a locale-specific suffix, and the new /LOCALE indicator within the FMTSEARCH= option specifies that the locale-specific catalog is to be used. For example, consider this code that creates a format called ANSWER, one for an English (US) locale and one for a French (France) locale:

```
options locale=en_us;

proc format locale library=work.myfmts;
     value answer 1='yes' 0='no';
     run;

options locale=fr_fr;

proc format locale library=work.myfmts;
     value answer 1='oui' 0='non';
     run;
```

The LIBRARY= values appear to be the same, but because the LOCALE option is given, the names of the catalogs are actually work.myfmts_en_us and work.myfmts_fr_fr. Here is how the catalogs are referenced with the FMTSEARCH= option:

```
options fmtsearch=(work.myfmts/locale);

data _null_;
     do x=0 to 2;
        put x= x=answer.;
        end;
     run;
```

If the LOCALE option value is en_US, the results will be:

OUTPUT 4: Output from the DATA step

```
x=0 x=no
x=1 x=yes
x=2 x=2
```

If the LOCALE option value is fr_FR, the results will be:

OUTPUT 5: Output from the DATA step

```
x=0 x=non
x=1 x=oui
x=2 x=2
```

The SAS code is the same, with no locale-specific information having to be provided, and the behavior is different based on the value of the LOCALE option.

DAY COUNT DIRECTIVE IN PICTURE FORMATS

If you need to indicate a day count in a picture format, you can do so with the %n directive when the DATATYPE= value is TIME. Here is an example:

```
/* Shows the %n directive for day count */
proc format;
     picture durtest(default=27)
       other='%n days %H hours %M minutes' (datatype=time);
     run;

data _null_;
     start = '01jan2010:12:34'dt;
     end = '15feb2010:18:36'dt;
     diff = end - start;
     put diff=durtest.;
     run;
```

OUTPUT 6: Output from the DATA step

```
diff=45 days 6 hours 2 minutes
```

DATETIME_UTIL DATATYPE VALUE

In the utility industry, a time value of midnight is often shown as 24:00:00 instead of 0:00:00. If you need this usage in a time format, you can use the DATATYPE= value of DATETIME_UTIL. It is otherwise identical to DATATYPE=DATETIME. For example:

```
proc format;
     picture abc (default=19)
          other='%Y-%0m-%0d %0H:%0M:%0S' (datatype=datetime_util);
     run;

data _null_;
     x = '01nov2008:00:00:00'dt; put x=abc.;
     x = '01nov2008:00:00:01'dt; put x=abc.;
     run;
```

OUTPUT 7: Output from the DATA step

```
x=2008-10-31 24:00:00
x=2008-11-01 00:00:01
```

FUNCTIONS IN LABEL SPECIFICATIONS

Likely the most significant feature added to PROC FORMAT in SAS 9.3 is the ability to use function names as labels. As with formats as labels, the function name appears in square brackets in place of a traditional quoted label. The function name is followed by parentheses to distinguish it from a format specification. For example:

```
proc format;
     value myfmt other=[myfunc()];
     run;
```

The specified function can be any function known to SAS, and the function should accept at most one argument. The function can accept either numeric or character arguments and return either numeric or character values. The argument that is passed to the function is the value being formatted. For example:

```
proc format;
     value myexmp '01nov2008'd - '30nov2008'd = [juldate7()]
                  other=[date9.];
     run;
data _null_;
     x='15nov2008'd; put x=myexmp.;
     x='01dec2008'd; put x=myexmp.;
     run;
```

The JULDATE7 function accepts a SAS date value and returns a Julian date with 4-digit years as a numeric value. The MYEXMP format will take care of converting the Julian date to a text string. So any SAS date value given that is in the month of November 2008 will be displayed as a Julian date, and all other SAS date values will use the DATE9. representation.

OUTPUT 8: Output from the DATA step

```
x=2008320
x=01DEC2008
```

Now let us consider some examples where you can create your own function using PROC FCMP. This is where the most extensive functionality can be found with this new feature.

Consider a simple function FTOC to convert Fahrenheit temperatures to Celsius and the converse function CTOF to convert Celsius temperatures to Fahrenheit. The results of the functions would be textual, with the numeric result, a degree sign, and the F or C as appropriate. Here is the PROC FCMP code to define those functions.

```
proc fcmp outlib=work.functions.smd;
   function ctof(c) $;
      return(cats(((9*c)/5)+32,'°F'));
   endsub;

   function ftoc(f) $;
      return(cats((f-32)*5/9,'°C'));
   endsub;
run;
```

Now that the functions have been defined, they can be used within a format with the function-as-label feature. For example:

```
options cmplib=(work.functions);
proc format;
     value ctof (default=10) other=[ctof()];
     value ftoc (default=10) other=[ftoc()];
     run;
data _null_;
     c=100; put c=ctof.;
     f=212; put f=ftoc.;
     run;
```

The results are as follows:

OUTPUT 9: Output from the DATA step

```
c=212°F
f=100°C
```

Note that in this example, we called the formats the same name as the functions used in the label. This is perfectly acceptable. Note also the use of the DEFAULT= option. You should always use a DEFAULT= option that refers to the desired default length because a length cannot otherwise be determined at format definition time.

In this example, we see that the function-as-label feature applies equally well to informats. In this case, we overcome the limitation that the TRAILSGN informat can support integer values only. The TSGN informat is introduced by invoking the TSGN function. The TSGN function in turn invokes the TRAILSGN informat and then divides by 100 and returns the numeric result, which is passed on to the TSGN informat.

```
proc fcmp outlib=work.functions.smd;

   function tsgn(text $);
      put 'in tsgn: ' text=;
      x = input(text,trailsgn10.);
      x = x/100;
      return(x);
   endsub;
```

```
run;

options cmplib=(work.functions);

proc format; invalue tsgn(default=10) other=[tsgn()];

data _null_;
     input x: tsgn.; put x=; cards;
1
1-
12-
123-
123+
1+
0
run;
```

The results are as follows:

OUTPUT10: Output from the DATA step

```
x=0.01
x=-0.01
x=-0.12
x=-1.23
x=1.23
x=0.01
x=0
```

So if you need a functionality beyond simple table lookup, you can provide an algorithmic process via PROC FCMP and the function-as-label feature.

CONTACT INFORMATION

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

Name:	Richard D. Langston
Enterprise:	SAS Institute Inc.
Address:	SAS Campus Drive
City, State ZIP:	Cary, NC 27513 USA
E-mail:	Rick.Langston@sas.com