PharmaSUG 2024 - Paper CC- 10025

Beyond the appearance: Unveiling the power of PROC FORMAT

Xiangyu Song, Daiichi Sankyo Company

ABSTRACT

The FORMAT procedure is designed for programmers to customize how raw data is read and written in SAS. However, beyond the common use of assigning descriptive labels to data values, the FORMAT procedure is also a strong tool in many other interesting applications. A well-designed SAS format can provide a powerful means to explore unexpected values and missingness in the data, tidy up unwanted variation in strings, perform calculations without creating additional variables, save computational resources for large dataset sub-setting and merging, and incorporate user defined SAS functions. In this paper, examples are provided to give a quick illustration of how efficient and reliable programming results can be achieved by creating user-defined informats/formats along with other SAS features such as the PICTURE statement, REGEX (Regular Expressions), and the FMCP procedure.

INTRODUCTION

PROC FORMAT is a procedure that creates variable labels, which are formally called informats and formats in SAS. Informats determine how raw data values are read and stored. Formats determine how variable values are printed. [1]

SAS provides a great variety of internal formats and informats, but still, there are situations in which none of those meet certain programming needs. The PROC FORMAT offers a practical solution here to close this gap. In addition, the user-defined informat and format have great potential in other aspects as well.

This paper includes examples demonstrating how PROC FORMAT can be used to:

- ✓ Identify data problems.
- ✓ Recoding and transforming messy data.
- ✓ Calculation and assigning value and style.
- ✓ Help to maintain a smarter program and enhance the running speed.
- ✓ Incorporate user defined SAS functions.

IDENTIFY DATA PROBLEMS

This technique is helpful for quickly obtaining a first glance at the dataset without making any modifications to it. Here is a basic example of inspecting the data points in a dataset using format. The SAS code following will show information about the SAShelp.Fish data and give relevant output.

Example 1: Identify the unexpected value and missing in the data

```
title 'The First Five Observations of Finland's Lake Laengelmaevesi
Fish Catch Data';
proc print data=sashelp.fish(obs=5);
run;
proc format;
value wt
low-0="Invalid"
0<-<1000="0-1000"
1000-high="1000+"
other="Unweighted";
run;

title 'Exploration of the variable weight';
proc freq data=sashelp.fish ;
table Weight/missing;
format Weight wt.;
run;
```

The First Five Observations of Finland's Lake Laengelmaevesi Fish Catch Data

Obs	SPECIES	WEIGHT	LENGTH1	LENGTH2	LENGTH3	HEIGHT	WIDTH
1	Bream	242	23.2	25.4	30.0	11.5200	4.0200
2	Bream	290	24.0	26.3	31.2	12.4800	4.3056
3	Bream	340	23.9	26.5	31.1	12.3778	4.6961
4	Bream	363	26.3	29.0	33.5	12.7300	4.4555
5	Bream	430	26.5	29.0	34.0	12.4440	5.1340

Exploration of the variable weight

The FREQ Procedure

WEIGHT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Unweighted	1	0.63	1	0.63
Invalid	1	0.63	2	1.26
0-1000	145	91.19	147	92.45
1000+	12	7.55	159	100.00

Use a data step to construct a new column using an IF statement subsequence to get the same result without requiring a user-defined format. Benefits of employing a user-defined format includes.

- ✓ Simple formats can conserve memory and avoid errors given there is no new datasets or new variables is created.
- ✓ The calling of format executes more quickly than SAS's IF statement and can directly use in most of PROCs.
- ✓ Changing the value thresholds of the exploration is easier and quick, would just need modifying the user-defined format's range.

This approach can be used to search for all the missing data in the entire dataset.

Example 1: Identify the unexpected value and missing in the data

```
proc format;
value $ miss
' '="Missing"
other="Not Missing";
value miss .
 ="Missing"
other="Not Missing";
run;

title 'Exploration of the Missing in Fish Catch Data';
proc freq data=sashelp.fish;
tables _numeric_ _character_/missing;
format _numeric_ miss.
_character_ $miss.;
run;
```

Exploration of the Missing in Fish Catch Data

The FREQ Procedure

WEIGHT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Missing	1	0.63	1	0.63
Not Missing	158	99.37	159	100.00

LENGTH1	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Not Missing	159	100.00	159	100.00

LENGTH2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Not Missing	159	100.00	159	100.00

LENGTH3	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Not Missing	159	100.00	159	100.00

HEIGHT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Not Missing	159	100.00	159	100.00

WIDTH	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Not Missing	159	100.00	159	100.00

SPECIES	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Not Missing	159	100.00	159	100.00

RECODING AND TRANSFORMING MESSY DATA

Controlling data readability, interpretability, and clarity is an essential piece of SAS programming task. As such, it is best practice to avoid unintentional variations within the same variable. This could be a challenge to use single technique to handle of potential consistencies.

Variation can be due to errors in spelling, discrepancies in uppercase or lowercase letters, differences in alignment and spacing, and other factors.

Potential Variations	
Spacing	Michael Smith » Michael Smith
Spelling differences	Center » Centre
Lower/Upper Cases	ANN » ann » Ann
Titles & Honorifics	Dr. Ann » Miss Ann » Ms. Ann » Ann
Abbreviations	Ave. » Avenue
Separators	Michael Smith » Michael,Smith
Formatting	777888999 » (777)-888-999

Starting from SAS 9.3, Regular Expression (REGEX) is supported in the INVALUE statement to create informat.

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/ REGEXPE) option is given after the quoted Perl regular expression to indicate that it is to be treated this way [2]

```
proc format;
invalue isnum (default=5) '/[0-9]/' (regexp) = _same_ other=_error_;
run;
```

The rules for regular expressions using the REGEXP option are the same as they are for the PRXPARSE function in the DATA step. The rules for the REGEXPE option are the same as they are for the PRXCHANGE function.

This feature provides a less painful solution for recoding the “messy” text variables. Example below shows a small data set with four variables: name, phone number, zip, and address with inconsistencies in every variable.

NAME	PHONE	ZIP	ADDRESS
Tony Hilll	(067) 672-7717	74303	45539 EWell Street
Devin Schaden	(178) 687-5010	35605	73743 Jay Ave.
Phillip ,Glover	(138) 516-9336	71726	2741 Raven Street
Duane, Kozey	454-372-7439	ZIP#42283	967Orn STREET
Dr. Roy Cruickshank	(441) 069-0749	16631	538, HILPERT ST
Tami Hahn	7674278018	33593-2345	82533 Carmen Avenue

Cleaning and standardizing of this data are highly desired to support further analysis.The code below shows how REGEX expression informats can standardize these variables efficiently.

Example 2: Perform data cleaning and standardization with REGEX informats

```
data example;
infile datalines delimiter="|";
length Name $ 30 Phone $20 Zip $10  Address $100 ;
input  Name $ phone $ Zip $ Address ;
datalines;
 Tony Hilll | (067) 672-7717 | 74303 | 45539 EWell Street
 Devin   Schaden | (178) 687-5010 | 35605 | 73743 Jay Ave.
 Phillip    ,Glover | (138) 516-9336 | 71726 | 2741 Raven Street
 Duane,  Kozey | 454-372-7439 | ZIP#42283 | 967Orn   STREET
 Dr. Roy Cruickshank | (441) 069-0749 | 16631 |  538, HILPERT ST
 Tami Hahn | 7674278018 | 33593-2345 | 82533 Carmen Avenue
run;

proc format ;
invalue $ zip(default=50) "s/(ZIP#?)?(\d{5})(.*)?/$2/i"
(REGEXPE)=_same_
other=_same_;
run;
data exmaple2;
type="OLD";
set example;
output;
type="NEW";
zip=input(zip,$zip.);
output;
```

```
run;
proc print data=exmaple2;
var type zip;
run;
```

Syntax details:

The REGEX informat .**"[s**② **/(ZIP#?)?(\d{5})(.*)?**③**/$2**④**/]i**①**"** contains 4 functional parts and can be read in the order labeled with ①②③④.

①The letter "i" at the end of the expression is the global option flag that modifies the behavior of searching. The flag "i" specifies the case insensitive searching. In this example, all case combinations of ZIP/zip/zIP... will be treated as a match.

② s/ is substitution operator, the syntax is "s/capturing/replacement/"

③ Within the first "/ /", (ZIP#?)?(\d{5})(.*)? **i**s the capturing pattern .

Parentheses () stand for groups in regex. Characters enclosing by parentheses () is treated as a single unit. Groups allows the backreference and reuse of the unit.

In the first group "(ZIP#?)?", question mark "?" stands for occurrence 0 or 1. Thus, (ZIP#?)? captures any of "zip" or "zip#" or nothing in the group 1.

In the second group (\d*) "\d" is s any digit character (0-9) and {5} stands for the occurrence of 5. Subsequent 5-digit characters are captured in group 2.

In the last capture group (. *) "." is a special character in REGEX. It means any character except newline. The last group captures anything after group 2 or nothing,

④ With the information captured in the ③, the only thing we need is to backreference to group 2 to reformat the ZIP code into 5-digit number as we wanted.

Below is the result of this transformation:

Obs	TYPE	ZIP
1	OLD	74303
2	NEW	74303
3	OLD	35605
4	NEW	35605
5	OLD	71726
6	NEW	71726
7	OLD	ZIP#42283
8	NEW	42283
9	OLD	16631
10	NEW	16631
11	OLD	33593-2345
12	NEW	33593

By applying a similar strategy, we may extend this approach to other variables in this dataset and eliminate variances in ZIP, Phone, and Name.

Example 2: Perform data cleaning and standardization with REGEX informats

```
proc format ;
/* REGEX informat of phone number  */
invalue $ phone (default=50)
```

```
"s/([^\d]*)(\d{3})([^\d]*)(\d{3})([^\d]*)(\d{4})/$2-$4-$6/i"
(REGEXPE) =_same_
other="ERROR";

/* REGEX informat of address      */
invalue $ address (default=50)
"s/(\d*)([,\s]s*)*(\w{1,})(\s*)(ST\w*)(\.)?/$1 \u\L$3\E Street/i"
(REGEXPE) =_same_
"s/(\d*)([,\s]s*)*(\w{1,})(\s*)(av\w*)(\.)?/$1 \u\L$3\E Avenue/i"
(REGEXPE) =_same_
other="ERROR";

/* REGEX informat of name      */
invalue $ fullname (default=50)
"s/(\s*\Dr\.\s*)?(\w{1,})(\W{1,})(\w{1,})/$4 $2/i" (REGEXPE)
=_same_;
run;

data exmaple2;
type="OLD";
set example;
output;
type="NEW";
phone=input(phone,$phone.);
address=input(address,$Address.);
name=input(name,$fullname.);
zip=input(zip,$zip.);
output;
run;
proc print data=exmaple2;
run;
```

Output produced by this code is here.

Obs	TYPE	NAME	PHONE	ZIP	ADDRESS
1	OLD	Tony Hilll	(067) 672-7717	74303	45539 EWell Street
2	NEW	Hilll Tony	067-672-7717	74303	45539 Ewell Street
3	OLD	Devin Schaden	(178) 687-5010	35605	73743 Jay Ave.
4	NEW	Schaden Devin	178-687-5010	35605	73743 Jay Avenue
5	OLD	Phillip ,Glover	(138) 516-9336	71726	2741 Raven Street
6	NEW	Glover Phillip	138-516-9336	71726	2741 Raven Street
7	OLD	Duane, Kozey	454-372-7439	ZIP#42283	967Orn STREET
8	NEW	Kozey Duane	454-372-7439	42283	967 Orn Street
9	OLD	Dr. Roy Cruickshank	(441) 069-0749	16631	538, HILPERT ST
10	NEW	Cruickshank Roy	441-069-0749	16631	538 Hilpert Street
11	OLD	Tami Hahn	7674278018	33593-2345	82533 Carmen Avenue
12	NEW	Hahn Tami	767-427-8018	33593	82533 Carmen Avenue

This paper is unable to cover all the specific details related to REGEX expression. If you would like to learn more about SAS's REGEX, please refer to the recommended reading area.

PERFORM CALCULATION AND CONDITONAL ASSIGNING STYLE OF NUMERIC VALUES

Most of the time, calculation and recoding for numeric variables are completed in the SAS Data step statement. However, this can also be achieved by utilizing a less aware format option PICTURE. A PICTURE format allows user to rescale the value and recode the style of the appearance of data in the user defined pattern. Symbols, units and can be easily customized even without creating additional variables. This is an example of performing rescaling and styling of the variable width using an income dataset which is available on Kaggle[3].

First five observations:

Obs	Occupation	All_workers	All_weekly	M_workers	M_weekly	F_workers	F_weekly
1	Public relations and fundraising managers	59	1557	24	Na	35	Na
2	Administrative services managers	170	1191	96	1451	73	981
3	Computer and information systems managers	636	1728	466	1817	169	1563
4	Financial managers	1124	1408	551	1732	573	1130
5	Compensation and benefits managers	23	Na	7	Na	16	Na

Variable information:

Occupation: Job title as given from BLS. Industry summaries are given in ALL CAPS.
All_workers: Number of workers male and female, in thousands.
All_weekly: Median weekly income including male and female workers, in USD.
M_workers: Number of male workers, in thousands.
M_weekly: Median weekly income for male workers, in USD.
F_workers: Number of female workers, in thousands.
M_weekly: Median weekly income for female workers, in USD.

Example 3: Data transformation and styling using picture format

```
Proc format;
picture number (default=20) ①
low-high="000,000,000,000 "②  (MULT=1000)③;
run;
proc print data=inc_occ_gender(obs=5);
format All_workers number.;
run;
```

Three key syntax of this FORMAT procedure.

① PICTURE statement

② `low-high="000,000,000,000 "` specify the pattern.

Note: A "0" allows SAS to omit leading zeros. A "9" in the picture forces the number to be displayed;

③ (MULT=1000) options tell SAS to perform the calculation to transfer number in thousands into individual account.

Note: Attention needs to be paid for the MULT= statement. This MULT= is not technically equal to the multiplication factor. The value specified in the MULT= is one of the two parameters of the actual factor for the calculation. The actual factor is 10 ^ (-[number of digits in the picture pattern])(MULT=X). Please always pay attention to your values.*

The outputs of this code:

Obs	Occupation	All_workers	All_weekly	M_workers	M_weekly	F_workers	F_weekly
1	Public relations and fundraising managers	59,000	1557	24	Na	35	Na
2	Administrative services managers	170,000	1191	96	1451	73	981
3	Computer and information systems managers	636,000	1728	466	1817	169	1563
4	Financial managers	1,124,000	1408	551	1732	573	1130
5	Compensation and benefits managers	23,000	Na	7	Na	16	Na

Unicode and escapchar are supported by PROC FORMAT.

Example 4: Unicode and Escapchar

```
proc format;
value $ na
"Na"="(*ESC*){unicode '2733'x}^{super na}";
run;
ods escapechar="^";
proc print data=inc_occ_gender(obs=6);
format All_workers number. All_weekly  M_weekly  F_weekly $na. ;
run;
```

The output will look like:

Obs	Occupation	All_workers	All_weekly	M_workers	M_weekly	F_workers	F_weekly
1	Public relations and fundraising managers	59,000	1557	24	✳na	35	✳na
2	Administrative services managers	170,000	1191	96	1451	73	981
3	Computer and information systems managers	636,000	1728	466	1817	169	1563
4	Financial managers	1,124,000	1408	551	1732	573	1130
5	Compensation and benefits managers	23,000	✳na	7	✳na	16	✳na
6	Human resources managers	254,000	1365	68	1495	186	1274

In additional, information of column style and color can be stored in SAS formats and directly read by PROCs.

Example 5: Assign style attributes in format

```
proc format;
value attr
low-10='^{style [color=VDABG font_size=9pt] less than 10k}'
10-100='^{style [color=MOBG  font_size=10pt] 10-100k }'
100-1000='^{style [color=VIBG   font_size=11pt] 100-1000k }'
1000-high='^{style [color=BIBG font_size=12pt] > 1000k}'
 ;
value colorattr
0-10='style=[background=cx82CA9D]'
10-100='style=[background=MOBG]'
100-1000='style=[background=VIBG]'
1000-high='style=[background=BIBG]';

value $  occup(default=200)
other=[propcase()];
run;
```

Output:

Obs	Occupation	All_workers	All_weekly	M_workers	M_weekly	F_workers	F_weekly
1	Public Relations And Fundraising Managers	59	1557	10-100k	Na	10-100k	Na
2	Administrative Services Managers	170	1191	10-100k	1451	10-100k	981
3	Computer And Information Systems Managers	636	1728	100-1000k	1817	100-1000k	1563
4	Financial Managers	1124	1408	100-1000k	1732	100-1000k	1130
5	Compensation And Benefits Managers	23	Na	less than 10k	Na	10-100k	Na

Occupation	All	Female	Male
Public Relations And Fundraising Managers	59	10-100k	10-100k
Administrative Services Managers	170	10-100k	10-100k
Computer And Information Systems Managers	636	100-1000k	100-1000k
Financial Managers	1124	100-1000k	100-1000k
Compensation And Benefits Managers	23	10-100k	less than 10k
Human Resources Managers	254	100-1000k	10-100k

Furthermore, in case there is any need, URL can also be integrated into the format.

Example 6: Create URL using format

```
data one;
input company $ x;
cards;
Google 200
SAS 5000
CDISC 200
;
run;

proc format;
value $urlfmt
'SAS'="http://www.sas.com/"
'Google'="http://www.google.com/"
'CDISC'="https://www.cdisc.org/";
run;



proc tabulate;
class company;
var x;
classlev company / style=[url=$urlfmt.];
table company,x;
run;
```

	X
	Sum
COMPANY	
CDISC	200.00
Google	200.00
SAS	5000.00

ENHANCE THE PERFORMANCE FOR LARGE DATASET PROCESSING.

Aside from helping programmers maintain a smaller and smarter dataset by avoiding creating additional variables, another very useful application of formats is to increase the efficacy of the program by performing value lookup and subset or merge datasets using the format.

The merging key or subsetting conditions can be stored in the SAS format and this will reduce the running time by ruling out redundant sorting steps and unnecessary IF statements.

Below is an example of large dataset merging using format.

Example 7: Table lookup and large dataset merging using format

```
%let _timer_start = %sysfunc(datetime());
data dm;
set sdtm.dm(keep=usubjid rfstdtc) end=last ;
length start $50 ;
fmtname='subject';
type = 'c';
start=usubjid;end=usubjid;
label =rfstdtc;
output;
if last then do;
HLO="O";
label="ERROR";
output;
end;
run;
proc format cntlin=dm;
run;
data lb;
set  lab;
rfstdtc=(put(usubjid,$subject.));
run;
data _null_;
  dur = datetime() - &_timer_start;
  put 30*'-' / ' TOTAL DURATION:' dur time13.2 / 30*'-';
run;
```

Syntax description：

The first data step of the program, FMTNAME, START, LABEL, and TYPE are created from the demographic datasets.

Then the CNTLIN option of PROC FORMAT allows us to create a format using the value created in the data step.

In the final data step, the RFSTDTC is mapped into LAB dataset

LOG

```
34          %let _timer_start = %sysfunc(datetime());
35          data dm;
36          set sdtm.dm(keep=usubjid rfstdtc) end=last ;
37          length start $50 ;
38          fmtname='subject';
39          type = 'c';
40          start=usubjid;end=usubjid;
41          label =rfstdtc;
The SAS System

42          output;
43          if last then do;
44          HLO="O";
45          label="ERROR";
46          output;
47          end;
48          run;

NOTE: There were 1798 observations read from the data set SDTM.DM.
NOTE: The data set WORK.DM has 1799 observations and 8 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.02 seconds


49          proc format cntlin=dm;
NOTE: Format $SUBJECT is already on the library WORK.FORMATS.
NOTE: Format $SUBJECT has been output.
50          run;

NOTE: PROCEDURE FORMAT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

NOTE: There were 1799 observations read from the data set WORK.DM.

51          data lb;
52          set  lab;
53          rfstdtc=(put(usubjid,$subject.));
54          run;

NOTE: There were 3359680 observations read from the data set WORK.LAB.
NOTE: The data set WORK.LB has 3359680 observations and 32 variables.
NOTE: DATA statement used (Total process time):
      real time           12.16 seconds
      cpu time            10.03 seconds


55          data _null_;
56            dur = datetime() - &_timer_start;
SYMBOLGEN:  Macro variable _TIMER_START resolves to 2036879490.26302
57            put 30*'-' / ' TOTAL DURATION:' dur time13.2 / 30*'-';
58          run;
```

```
------------------------------
 TOTAL DURATION:   0:00:12.24
------------------------------
```

Let's compare the total processing time of this approach with the data step merge and proc sql
Data step merge

```
/**2) Data Step Merge ************/
%let _timer_start = %sysfunc(datetime());
proc sort data=sdtm.dm(keep=usubjid rfstdtc) out=dm;
by usubjid;
run;
proc sort data=lab out=lb;
by usubjid;
run;
data all;
merge dm(in=a)  lab(in=b);
by usubjid;
if  b ;
run;
data _null_;
  dur = datetime() - &_timer_start;
  put 30*'-' / ' TOTAL DURATION:' dur time13.2 / 30*'-';
run;
```

```
62         proc sort data=sdtm.dm(keep=usubjid rfstdtc) out=dm;
63         by usubjid;
64         run;

NOTE: There were 1798 observations read from the data set SDTM.DM.
NOTE: The data set WORK.DM has 1798 observations and 2 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.03 seconds
      cpu time            0.02 seconds


65         proc sort data=lab out=lb;
66         by usubjid;
67         run;

NOTE: There were 3359680 observations read from the data set WORK.LAB.
NOTE: The data set WORK.LB has 3359680 observations and 31 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           24.77 seconds
      cpu time            27.84 seconds


68         data all;
69         merge dm(in=a) lab(in=b);
70         by usubjid;
71         if  b ;
72         run;
```

```
NOTE: There were 1798 observations read from the data set WORK.DM.
NOTE: There were 3359680 observations read from the data set WORK.LAB.
NOTE: The data set WORK.ALL has 3359680 observations and 32 variables.
NOTE: DATA statement used (Total process time):
      real time           11.93 seconds
      cpu time            9.15 seconds


73         data _null_;
74           dur = datetime() - &_timer_start;
SYMBOLGEN:  Macro variable _TIMER_START resolves to 2036879502.50149
75           put 30*'-' / ' TOTAL DURATION:' dur time13.2 / 30*'-';
76         run;

The SAS System

------------------------------
 TOTAL DURATION:   0:00:36.76
------------------------------
```

Proc SQL Join

```
/**3) look up by SQL************/
proc sql;
create table all
as select a.* from
lab a
left join sdtm.dm(keep=usubjid rfstdtc)  b
on a.usubjid=b.usubjid;
quit;
```

```
78         %let _timer_start = %sysfunc(datetime());
79         proc sql;
80         create table all
81         as select a.* from
82         lab a
83         left join sdtm.dm(keep=usubjid rfstdtc)  b
84         on a.usubjid=b.usubjid;
NOTE: Table WORK.ALL created, with 3359680 rows and 31 columns.
85         quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           38.65 seconds
      cpu time            39.89 seconds


86         data _null_;
87           dur = datetime() - &_timer_start;
SYMBOLGEN:  Macro variable _TIMER_START resolves to 2036879539.26603
88           put 30*'-' / ' TOTAL DURATION:' dur time13.2 / 30*'-';
89         run;

------------------------------
 TOTAL DURATION:   0:00:38.66
------------------------------
NOTE: DATA statement used (Total process time):
```

```
      real time             0.00 seconds
      cpu time              0.00 seconds
```

The total time for these three is summarized below:

	PROC Format and PUT	DATA Step and Merge	PROC SQL
Total	**12.24**	**36.76**	**38.66**

In this example, approach 1 can save around 20 seconds for a left merging task.

CREATE FUNCTIONS WITH PROC FCMP

PROC FORMAT is able to use function names as labels.

The base syntax is

```
proc format;
  value fmt
  other=[func()];
run;
```

The specified function can be any SAS function that accepts no more than one argument known to SAS. Below is an example that gives a quick illustration of how to fit the code into this syntax

```
 proc format;
  value date_ext
  other=[datepart()];
  run;
  data _null_;
  x = '31may2024:00:00:00'dt;  put x=date_ext.;
  run;
```

The SAS log writes the X as

```
x=23527
```

User defined function created by PROC FCMP can be used in the format as well.

This allows for more complex calculations. Here is an example to create a format to calculate factorial results of the number [4].

Example 8: Format produces factorial results

```
proc fcmp outlib =work.function.factor;
 function factor(value);
 if (value =0)
 then return(1);
 else return(Value * factor (Value - 1));
  endsub;
 quit;
proc format;
  value fact
  other=[factorl()];
  run;
options cmplib=work.function;

data exmaple;
 do number=1 to 9;
 factorial=put(number,fact.);
 output;
 end;
 run;
proc print;run;
```

Output is in the Figure 6

Obs	NUMBER	FACTORIAL
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880

The use of function formats can be further increased by integrating with the REGEX feature, which was covered in the previous section. Here's an illustration of a user-defined informat $dates can conditionally alter variations of dates based on the pattern of input values.

Example 9: Wide range date transformation format

```
data messydates;
infile datalines delimiter="," dsd;
length date $20.;
input date $;
cards;
2024/01- 02,
2024  01UN,
202401 02  ,
02 JAN2024,
UNjan/2024,
2024Jan,
2024-02-31,
2024UNKUN,
01.02.2024,
2024JAN2,
01-02-2024,
2024UNK02,
2023FEB29,
2024-02-29
;
run;
options cmplib=work.function;
data cleandates;
set messydates;
date_new=input(date,$dates.);
run;
```

Output;

Obs	date	date_new
1	2024/01- 02	2024-01-02
2	2024 01UN	2024-01
3	202401 02	2024-01-02
4	02 JAN2024	2024-01-02
5	UNjan/2024	2024-01
6	2024Jan	2024-01
7	2024-02-31	Invalid
8	2024UNKUN	2024
9	01.02.2024	2024-01-02
10	2024JAN2	2024-01-02
11	01-02-2024	2024-01-02
12	2024UNK02	2024---02
13	2023FEB29	Invalid
14	2024-02-29	2024-02-29

The detailed code is in below:

```
proc fcmp outlib=work.function.example;
function yyyymmdd(dvar $) $;
dvar_=compress(dvar,' K#!$%&()*+-./:;<=>?[\]^_{|}~)');
yy=substr(dvar_,1,4);
mm=substr(dvar_,5,2);
if mm in ("UNK" "UN") then do;
mm="-";
end;
dd=substr(dvar_,7,2);
if dd in ("UN") then do;
dd="";
if mm in ("-") then do;
mm="";
end;
end;
length cc $20;
cc=catx('-',yy,mm,dd);
return(cc);
endsub;

function mmddyyyy(dvar $) $;
length cc $20;
dvar_=compress(dvar,' K#!$%&()*+-./:;<=>?[\]^_{|}~)');
yy=substr(dvar_,5,4);
mm=substr(dvar_,1,2);
if mm in ("UN") then do;
mm="-";
end;
dd=substr(dvar_,3,2);
if dd in ("UN") then do;
dd="";
if mm in ("-") then do;
mm="";
end;
end;
length cc $20;
```

```
cc=catx('-',yy,mm,dd);
return(cc);
endsub;


function ddmonyy(dvar $) $;
dvar_=strip(upcase(compress(dvar,' K#!$%&()*+-./:;<=>?[\]^_{|}~)')));
length cc dvar_ $20;
if  length(dvar_)<8 or substr(dvar_,1,2) in ("UN") then do;

cc=substr(put(input("01"||substr(dvar_,3,3)||substr(dvar_,6,4),date9.
),e8601da.),1,7);
 end;
if  length(dvar_)=8 then do;
     dvar_="0"||dvar_;
    end;

if ( length(dvar_)=9  and substr(dvar_,1,2) ^= ("UN")) then do;
   dt=input(dvar_,date9.);
   if dt=. then do;
   cc="Invalid";
   end;
    if dt ne . then do;
     cc=put(dt,e8601da.);
   end;
end;
return(cc);
endsub;

function yymondd(dvar $) $;
length cc $20;
dvar_=strip(upcase(compress(dvar,' K#!$%&()*+-./:;<=>?[\]^_{|}~)')));

if  length(dvar_)<8 or substr(dvar_,length(dvar_)-1,2) in ("UN")
then do;

cc=substr(put(input("01"||substr(dvar_,5,3)||substr(dvar_,1,4),date9.
),e8601da.),1,7);
 end;
 if  length(dvar_)=8 then do;
     dvar_=substr(dvar_,1,7)||"0"||substr(dvar_,8,1);
    end;

if  length(dvar_)=9  and substr(dvar_,length(dvar_)-1,2) ^= ("UN")
then do;

dt=input(substr(dvar_,8,2)||substr(dvar_,5,3)||substr(dvar_,1,4),dat
e9.);
   if dt=. then do;
   cc="Invalid";
   end;
    if dt ne . then do;
     cc=put(dt,e8601da.);
   end;
end;
return(cc);
endsub;
```

```
run;
run;


proc format ;
 invalue  $ dates (default=20)
 /* Any YYYY MM DD */
 "/^[0-9]{4}[^\w]*(((0?[13578]|(10|12)|(UN|UNK))[^\w]*([1-2][0-
9]|3[0-1]|0[1-9]|UN))|(02[^\w]*([1-2][0-9]|0?[1-
9]$|UN))|((0?[469]|11|UN|UNK)[^\w]*([1-2][0-9]|0?[1-9]|30|UN)))/"
(REGEXP) =[yyyymmdd()]

 /* Any MM DD YYYY*/
"/^(((0?[13578]|(10|12)|(UN|UNK))[^\w]*([1-2][0-9]|3[0-1]|0[1-
9]|UN))|(02[^\w]*([1-2][0-9]|0?[1-
9]$|UN))|((0?[469]|11|UN|UNK)[^\w]*([1-2][0-9]|0?[1-
9]|30|UN)))[^\w]*[0-9]{4}/" (REGEXP)=[mmddyyyy()]

 /* Any DD Monthname YYYY*/
"/^(0?[1-9]|[12][1-
9]|3[01]|UN)?[^\w]*(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
[^\w]*[0-9]{4}/i"
(REGEXP)=[ddmonyy()]
 /* Any YYY Monthname DD*/
"/^[0-
9]{4}[^\w]*(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[^\w]*(0?
[1-9]|[12][1-9]|3[01]|UN|$)/i"
 (REGEXP)=[yymondd()]

 /* Year only*****/
"/^[0-9]{4}$/" (REGEXP) =_same_

other="Invalid"

;
run;
```

REFERENCES

[1] PROC FORMAT: Overview: FORMAT Procedure. . Im Internet: https://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#format-overview.htm; Stand: 24.06.2024

[2] SAS Help Center: Creating an Informat Using Perl Regular Expressions. . Im Internet: https://go.documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/n1jriq5xib5j45n1pwpwzk311v0p.htm; Stand: 03.07.2024

[3] Predict Online Dating Matches Dataset. . Im Internet: https://www.kaggle.com/datasets/rabieelkharoua/predict-online-dating-matches-dataset; Stand: 03.07.2024

[4] McNeill B. Paper SAS2125-2018: FCMP: A Powerful SAS® Procedure You Should Be Using.

RECOMMENDED READING

Windham KM. Introduction to Regular Expressions in SAS.

RegExr: Learn, Build, & Test RegEx. RegExr. Im Internet: https://regexr.com/; Stand: 03.07.2024