

Formats As A Programming Tool

John R. Gerlach, Maxim Group; Plymouth Meeting, PA

Alan Rinkus, Fleet Financial Group; Horsham, PA

Abstract

The Format procedure in SAS® provides the means to create user-defined informats and formats for character and numeric data that facilitate reading and writing data, respectively. Besides its typical use, however, this procedure offers much as a programming tool. For example, the Format procedure can be used for doing table look-ups, performing data validation, and subsetting a data set. Also, this procedure has the capability of creating formats, even dynamic formats, directly from SAS data sets. This paper gives a brief overview of primary features and discusses programming aspects of this powerful, under-utilized procedure.

An Analogy for the Novice

Suppose you have a SAS data set *survey* that contains the variables *ms* and *inc*. The categorical variable *ms* has values S,M,W, and D and the variable *inc* denotes a continuous numeric representing dollar amounts. Obviously, for reporting purposes, these variable identifiers and their values are not readily discernible. So, we assign a label to the variables, indicating that the variable *ms* denotes 'Marital Status' and the variable *inc* denotes 'Yearly Income.' Also, we employ appropriate formats that will convert the values using a user-defined format called *\$msf* for the character variable *ms*, and the SAS-supplied format *dollar8.* for the numeric variable *inc*.

```
libname proj 'r:\proj01';
libname library 'r:\proj01';

proc format library=library;
  value $msf 'S' = 'Single'
            'M' = 'Married'
            'W' = 'Widowed'
            'D' = 'Divorced';
run;

proc datasets library=proj;
  modify survey;
  label ms = 'Marital Status'
        inc = 'Income';
  format ms $msf. inc $dollar8.;
quit;
```

Consequently, the variable identifiers and their values have been enhanced using labels and formats.

Standard Informats & Formats

SAS provides a rich collection of standard informats and formats used for reading and writing character, numeric, and datetime values, several of which are listed below.

Several SAS supplied informats:

w.	Standard numeric
lbw.d	Integer binary
OCTALw.d	Converts positive octal values to integers
PERCENTw.	Converts percentages to numeric values
DATEw.	Dates of the form DDMMYY
DDMMYYw.	Date values
JULIANw.	Julian date (e.g., YYDDD)
MONYYw.	Month and year
\$w.	Standard character
\$CHARw.	Characters with blanks
\$EBCDICw.	EBCDIC to ASCII
\$VARYINGw.	Varying-length char values

Several SAS supplied formats:

w.	Standard numeric
w.d	Standard numeric
BESTw.	SAS chooses best notation
COMMAw.d	Commas in numbers
DOLLARw.d	Dollar sign & commas
DATEw.	Dates of the form DDMMYY
JULIANw.	Julian date (e.g., YYDDD)
MONYYw.	Month and year
\$w.	Standard character
\$CHARw.	Characters with blanks
\$EBCDICw.	EBCDIC to ASCII
\$VARYINGw.	Varying-length char values

Besides having access to many standard SAS informats and formats, you can create your own user-defined informats and formats using the Format procedure.

User-defined Informats & Formats

User-defined informats convert input values to a different form for processing data; whereas, a user-

defined format converts a value to a different form for output.

User-defined informats read only character (text) data. That is, such informats can convert character data into real numeric values; but, they cannot convert real numbers into characters. User-defined formats always associate variable values with character values. Thus, even formatted numeric values are character numbers, which explains why the SAS System writes a warning message when performing an arithmetic operation on a user-formatted numeric value.

The Format procedure creates two kinds of formats: value formats, which convert output values into a different form; and, picture formats that specify templates for printing numbers. Only numeric variables can have picture formats associated to them.

Once created, these items are associated with variables in several ways:

- Using an INPUT, PUT, ATTRIB, INFORMAT, or FORMAT statement in a DATA step, accordingly.
- Using an INFORMAT or FORMAT statement in a PROC step, accordingly.

Note: User-defined formats and informats are stored as entries in a SAS catalog, not as part of a SAS data set. Thus, permanently associating a user-defined format or an informat with a variable does not mean that they become a part of the data set. In fact, the format catalog need not reside even in the same data library as the data set. Thus, complications will arise if you do not have access to a format or informat that has been associated with a data set variable.

The Format Procedure

This paper includes a syntax diagram of the FORMAT procedure, which is intended to be a reference only.

```
PROC FORMAT CNTLIN = SAS-data-set
  CNTLOUT = SAS-data-set
  FMTLIB
  LIBRARY = libref<.catalog-name>
  MAXLABELLEN = n
  MAXSELEN = n
  NOREPLACE NOTSORTED PAGE;

  VALUE name <(format-option-list)>
    range='formatted-value-1'
    <...range-n='formatted-value-n'>;

  PICTURE name <(format-option-list)>
    range-1='picture-1'
    <(picture-option-list)>
    <...range-n='picture-n'>
    <(picture-option-list-n)>;
  INVALUE name <(informat-option-list)>
    <'>range1<'> =informatted-
    value-1 <...<'>
    range-n<'>=informatted-value-n>;
```

```
SELECT entry_list;
EXCLUDE entry_list;
RUN;
```

Rather than discuss the syntax diagram of the Format procedure (too boring for words), the code below illustrates how to create user-defined informats and formats.

```
proc format;
  value sexf 1 = 'Male'
            2 = 'Female';

  value $msfmt 'S' = 'Single'
              'M' = 'Married'
              'W' = 'Widowed'
              'D' = 'Divorced';

  value $dxif '4019' = 'Essential Hypertension'
            '25000' = 'Diabetes Mellitus'
            '7807' = 'Malaise & Fatigue'
            '78650' = 'Chest Pain, NOS'
            other = 'Other Diagnoses';

  invalue gradef 'A' = 4      'B' = 3
                'C' = 2      'D' = 1
                'F' = 0;

  invalue $quest 'A' = 'E'    'B' = 'D'
                'C' = 'C'    'D' = 'B'
                'E' = 'A';

  picture thous 0 - high = '00,009K'
    (mult=.001);

  picture dolr low <- 0 = '000,009.00'
    (prefix = '$-')
    0 - high = '000,009.00'
    (prefix = '$');
```

run;

The rest of this paper discusses the Format procedure as a programming tool.

Reporting on Missing / Non-Missing Values

When doing data analysis, it is often useful to know whether variables contain missing values. Rather than produce a frequency distribution listing all values, including missing values, it would be better to produce a two-level distribution indicating missing and non-missing values. You can obtain this report by creating two formats, for numeric and character data, then using the Freq procedure with the Missing option of the Tables statement.

Consider the Format procedure below that creates the two user-defined formats, called `missf` and `$missf`. Notice how the missing value (.) and blank along with the Other option of the VALUE statement dichotomize any numeric or character variable.

```
proc format;
  value missf . = 'Missing'
              other = 'Missing';
  value $missf ' ' = 'Missing';
```

```

other = '^Missing';
run;

```

To finish off this programming tool, we define a macro that utilizes the formats accordingly. Thus, by specifying the name of the data set, the desired numeric and character variables (all variables by default), the macro produces a frequency distribution for each variable indicating two levels: missing and non-missing values.

```

%macro missrep(ds,nvar=_numeric_,
  cvar=_character_);
  proc freq data=&ds.;
    tables &nvar. &cvar. / missing;
    title "Missing Values Report";
    title2 "Data Set %upcase(&ds.)";
  run;
%mend missrep;

%missrep(proj.survey, nvar=inc, );

```

Performing Data Validation

Proper data analysis requires well-defined data that have been validated. Often the quest for validated data becomes a nasty, tedious project consuming much resources. Considering that most validation checks can be written as Boolean expressions, it's possible that these checks can be stored in a user-defined format. Also, the format becomes *de facto* documentation of the validation process. Below is a syntax diagram that illustrates how to create the validation checks using Proc Format.

```

proc format;

  value Name of format containing QC checks
    1 = "Boolean expression"
    2 = "Boolean expression"
    :
    :
    n = "Boolean expression" ;

run;

```

Now, consider a clinical trial study consisting of several protocols representing hundreds of patients and lots of data, including: demographics, lab tests, vital signs, and dosing, each represented by its own SAS data set. The following code defines several formats that contain validation checks intended for a specific data set, such as one representing demographic data.

```

proc format;

  value qcpat
    1 = "gender not in('M','F')"
    2 = "age gt 24 | age eq ."
    3 = "1 le wt le 35"
    4 = "ht eq ."
    5 = "gender eq 'M'";

  value qclab
    1 = "syst gt 200 and dias gt 150"
    2 = "syst eq . or dias eq .";

  value qcdose
    1 = "date eq ."
    2 = "dose not in (3,6,12,25)";

```

```
run;
```

The %qcrep macro, written below, performs the validation process by utilizing these formats. The following examples illustrate proper use of the %qcrep macro.

```
%qcrep(prot021.patient, qcpat)
```

Validates the data set *patient* using the *qcpat* format, which resides in the WORK library because of the default value of the *qclib* parameter. Also, the macro uses all the variables in the patient data set and generates a detailed report, as well.

```
%qcrep(prot054.lab, qclab, qclib=prot021,
  qcvar=pid vdate syst dias, qcdet=N)
```

Validates the data set *lab* using the *qclab* format, which resides in a different library. Also, the macro uses only several variables from the data set and does not generate a detailed report.

```
%qcrep(prot054.dose, qcdose, qclib=prot021,
  qcvar=pid date dose, qcdet=Y)
```

Validates the data set *dose* using the *qcdose* format, which resides in a different library. Also, the macro uses only several variables from the data set and generates a detailed report.

```

%macro qcrep(qcdset,qcfmt,qclib=work,
  qcvar=_all_,qcdet=Y);

  options number pageno=1 date;

  proc format library=&qclib.
    cntlout=qcchks(keep=start);
    select &qcfmt.;
  run;

  data _null_;
    set qcchks nobs=n;
    if _n_ eq 1
      then call symput('qcchks',
        left(put(n,2.)));
    call symput('qcchk' ||
      trim(left(put(_n_,2.))),
      left(put(_n_,&qcfmt..)));
  run;

  data qcrep;
    set &dsn.(keep=&var.);
    %do i = 1 %to &qcchks.;
      if &&qcchk&i..
        then do; qcchk = &i.; output; end;
    %end;
  run;

  proc freq data=qcrep noprint;
    tables qcchk / out=qccnts;
  run;

  title1 "QC Analysis of %upcase(&qcdset.)";
  proc print data=qccnts noobs label;
    var qcchk count;
    format qcchk &qcfmt.. count comma8.;
    label qcchk = 'QC Check'
      count = '#';

```

```

title2 "Frequency of QC Checks" ;
run;

%if %upcase(&qcdet.) eq Y
%then %do;
proc sort data=qcrep;
by qcchk;
run;
proc print data=qcrep label noobs;
by qcchk;
format qcchk &qcfmt.;
title2 "Detailed Listing" ;
run;
%end;
%mend qcrep;

```

Subsetting Data

Assume, for a study population, you need to obtain selected lab tests occurring after a prescription date. Unfortunately, the lab tests contain only verbatim information. Thus, the initial query, based on patient ID and prescription date contains extraneous lab tests.

```

proc sql;
create table thertest as
select l.emrptid, l.test, l.result
from cohort c, ordb.labtest l
where c.emrptid eq l.emrptid
and l.labdate gt c.rxdate
order by l.emrptid, l.labdate;
quit;

```

Fortunately, there is another source, a 'bridge' file that maps all unique verbatim terms to a code value based on the Physician's Current Procedure Terminology (CPT) manual, from which we extract only the CPT codes of interest. An excerpt of the file follows.

CPT4 Class Code / Verbatim Terms

```

78350 *MISC - BONEDENSI
78350 *X-RAY - BONE DENSITOMETRY
78350 *X-RAY - BONE DENSITY
78350 *X-RAY - BONEDENS-MCT
78350 |BONE DENSITOMETRY - DEXA
78350 09/30/96 BONE MINE
78350 BENE DENSITY SCREENING
78350 BMD/DRU
78350 BMD/PR
78350 BMD/PRU
78350 BOME DENSITY
78350 BONE DENIS

```

In order to obtain only selected lab tests, we first read the file containing the CPT codes of interest and create a user-defined format that will facilitate the subsetting process, illustrated below.

```

filename labs 'r:\proj01\labs.dat';

data labtstf;
length start label $60;
retain fmtname 'LABTSTF' type 'C' label;
infile labs firstobs=3
delimiter='05'x missover;
input label $5. @7 start &$;
keep fmtname start type label;

```

```
run;
```

The following DATA step creates an observation that will represent the OTHER level as needed in the VALUE statement. This ensures that the format will handle all possible values in the verbatim term. That is, when the format cannot convert the verbatim term to a respective CPT code, then the formatted value is blank. The subsequent DATA step concatenates the two data sets LABTSTF and OTHER, thereby creating the appropriate control data set.

```

data other;
length start label $40;
retain fmtname 'LABTSTF' type 'c';
start=' '; label=' '; hlo = 'O';
run;

data labtstf;
set labtstf other;
run;

```

Once the data set contains all pertinent information needed, the CNTLIN option of the FORMAT procedure processes the data set and creates the format called LABTSTF.

```

proc format cntlin=labtstf maxselen=60;
run;

```

Then, the subsetting problem becomes trivial in that the DATA step asks the question: Does the verbatim term match to a CPT code of interest? If not, toss the observation.

```

data labtst;
set thertest;
if put(labtst,$labtstf.) eq ''
then delete;
run;

```

Optimizing Precision

Suppose you need to produce a flat file that must preserve the precision of the numeric data, which might otherwise impair the analysis. Using the BEST12 format produces an ugly file, as well as loses information with respect to precision. Performing univariate analysis on the numeric variables seems impractical and fails to reveal enough information about precision in order to specify an appropriate numeric format.

A robust solution would be to determine the precision of each numeric variable and, then, to create a CNTLIN data set that contains all the picture formats needed to write the numeric data preserving their original precision. In fact, the following codes creates as many picture formats as needed, one for each numeric variable.

The following SQL step creates a macro variable that denotes the number of numeric variables stored in the data set.

```

proc sql noprint;
select count(*) into: nvars

```

```

from dictionary.columns
where memname eq "&dsn." and
  substr(type,1,1) eq 'n';
quit;

```

```

%let nvars = %sysfunc(
  trim(%sysfunc(left(&nvars.))));

```

The following DATA step determines the highest precision of each numeric variable (of similar magnitude) and creates a single observation data set that contains all the patterns that become the picture formats needed.

```

data pats;
  retain to '000000000' from '123456789';
  set &dsn.(keep=_numeric_) end=eof;
  array nvars{*} _numeric_;
  array pat{*}$12 pat1-pat&nvars.;
  retain pat1-pat&nvars.;
  do i = 1 to &nvars.;
    tmp = left(put(nvars{i},best12.));
    if (length(tmp) gt length(pat{i}))
      or pat{i} eq ''
      then pat{i} =
        translate(tmp,to,from);
  end;
  if eof then output;
  drop i tmp;
run;

```

The final DATA step creates the control data set that contains the information need to create as many formats as there are numeric variables, each ensuring optimal precision.

```

data picfmts;
  length fmtname $8 label $12 nvar $7;
  retain start 'LOW' end 'HIGH'
    hlo 'LH' type 'P';
  set pats;
  array nvars{*} _numeric_;
  array pat{*}$12 pat1-pat&nvars.;
  do i = 1 to &nvars.;
    call vname(nvars{i},nvar);
    fmtname = compress(nvar) || 'F';
    label = pat{i};
    output;
  end;
  keep fmtname type start end hlo label;
run;

```

The resulting control data set, used to create the format(s), contains six variables: *fmtname*, *type*, *start*, *end*, *hlo*, and *label*. The variables *start*, *end*, *hlo*, and *type* indicate the range, from the lowest to the highest value, and the type of format, namely, a picture format used specifically for writing numeric data. The RETAIN statement assigns these constant variables their appropriate values.

Using the CNTLIN option of the FORMAT procedures allows you to create multiple formats from a single SAS data set; thus, the code below accomplishes the task. The FMTLIB option produces a listing of all the formats just created.

```

proc format cntlin=picfmts ffmtlib;

```

```

run;

```

Producing Better Titles

When doing By-group processing, it is possible to enhance the titles using the #byvar and the #byval specifiers. Using labels and formats makes the title statement even more informative. Consider the following SAS code that utilizes formats for enhancing both output and the title statement.

```

proc format;
  value $prod 'G' = 'Gold'
              'P' = 'Platinum'
              'S' = 'Standard';

run;

options nobyline;

proc print label;
  id group;
  var exdt oa da;
  by prod;
  format prod $prod.
        exdt date7. oa da comma8.;
  label prod = 'Product'
        group = 'Group'
        exdt = 'Extract Date'
        oa = '# Open Account'
        da = '# Debit Active';
  title1 " Frequency of Accounts" ;
  title3 " #byvar(prod) (#byval(prod))" ;
run;

```

Imputing Data

In the credit card business, many companies solicit information from various credit bureaus (e.g., TRW) about prospective clients. The bureau that supplies vital information depends on what part of the United States and the specific sales campaign. In short, the bureau code is a function of two variables: the SCF code, derived from the zip code, and the specific campaign. That is, the converted value of the SCF depends on the specific campaign.

Using dynamic formats in SAS makes it possible to impute the bureau code as a function of two variables. Consider the following code.

```

proc format;
  value $jan97f '006' = 'TRW'
              '007' = 'TRW'
              '010' = 'TU'
              '030' = 'TRW';

  value $jun97f '006' = 'TU'
              :
              '030' = 'TRW';

  value $jun97f '006' = 'TRW'
              :
              '030' = 'TRW';

  value $campf 'JAN97' = '$JAN97F'

```

```
'JUN97' = '$JUN97F'
      :
      :
'JAN98' = '$JAN98F';

run;
```

The user-defined character formats above provide the necessary information to impute the bureau code as a function of a geographic component and the specific sales campaign. The PUTC function utilizes a character format at run time, which denotes the campaign, and determines how the dynamic format converts the value of SCF, thereby imputing the bureau code.

```
data test;
  set camps(keep=campaign zip);
  scf = substr(zip,1,3);
  camp = put(campaign,$campf.);
  burcod = putc(scf,camp);
run;

proc freq;
  tables burcod / missing;
  title 'Distribution of Bureau Codes';
run;
```

Recoding Data

In lieu of a standard value, such as normal temperature, a special character (*) supplants the actual value in order to save time. However, when doing the analysis, the special character must be converted into its true value, which poses a programming task that includes an IF statement in a DATA step. An easier solution utilizes an enhanced numeric format, as follows.

```
proc format;
  invaluenormt '*' = 98.6
              other = _same_;
run;
```

Then, when reading the raw data, simply use the informat in the INPUT statement, as follows.

```
data;
  infile cohort;
  input pid $9. bp $6. temp normt.;
run;
```

Another situation involves a marketing survey that contains three sections of questions such that the scaled responses have different meaning and order. For example, Section #1 lists the responses in the order: Extremely Often, Frequently, Occasionally, Rarely and Very Rarely; using the letters A through E, respectively. However, Section #2 lists the responses in reverse order; and, Section #3 lists the responses in a different order from the others.

```
proc format;
  value sectf 1 = '$sect1f'
             2 = '$sect2f'
             3 = '$sect3f';

  invaluen $sect1f 'A' = 'Extremely Often'
                  'B' = 'Often'
                  'C' = 'Occasionally'
                  'D' = 'Rarely'
                  'E' = 'Very Rarely';
```

```
invaluen $sect2f 'A' = 'Very Rarely'
                :
                :
                'D' = 'Often'
                'E' = 'Extremely Often';

invaluen $sect3f 'A' = 'Occasionally'
                'B' = 'Often'
                'C' = 'Rarely'
                'D' = 'Extremely Often'
                'E' = 'Very Rarely';

run;
```

Here the user-defined character informats \$sect1f, \$sect2f, \$sect3f and the numeric format sectf provide the necessary information to convert the response data, having values A through E, according to its section in the survey.

```
data survey;
  set survey;
  sect = put(section,sectf.);
  resp = inputc(response, sect);
run;

proc freq;
  tables resp;
  title 'Distribution of Responses';
run;
```

Using Formats As Labels

When working with longitudinal data, assume you wish to print the actual dates for a specific time period and different text depending on other time periods. Using a standard SAS date format as a label and quoted text for the other ranges achieves this task. Consider the following SAS code that uses an existing SAS date format as a label denoting a specified time period, and descriptive text otherwise.

```
proc format;
  value titratf
    '01jan95'd - '31dec95'd = 'Screening'
    '01jan96'd - '30jun97'd = [date7.]
    '01jul97'd - '31dec97'd = 'Post Study'
    other = 'Error';
run;
```

Improving Efficiency

By default, the Format procedure organizes the range values of a format in ascending order in order to use a binary tree search when locating formatted values. Sometimes, however, that poses an efficiency issue when your range values do not comply to a collating sequence, but rather to the likelihood of occurrence of a value. For instance, given a large cohort of patients each having numerous diagnoses, it would be more efficient to obtain the frequency of those ICD-9 codes, then implement the format such that the most frequent diagnoses are found immediately. Using the NOTSORTED option specifies that the SAS System will store ranges for formats and informats in the order that you define them. For our

purposes, however, we use a control data set with the added HLO attribute that will accomplish the same task.

```
proc freq data=cohort;
  tables dxcode / noprint
    out=dxcf(keep=dxcode count);
run;

proc sql;
  create table dxcf as
  select 'dxcf' as fmtname, 'C' as type,
    dxcf.dxcode as start, dxname as label,
    'S' as hlo
  from dxcf, dxnames
  order by count desc;
quit;

proc format cntlin=dxcf fmlib;
run;
```

Conclusion

The Format procedure offers extraordinary possibilities for the SAS user, far beyond its fundamental use as a utility procedure. Also, this paper should convince the reader to recognize this procedure as a programming tool that greatly facilitates many tasks.

Author Information

John R. Gerlach
183 Freeland Drive
Collegeville, PA 19426
610.454.9967

Alan Rinkus
Fleet Credit Card Services
Horsham, PA
215.444.3769

SAS is a registered trademark of SAS Institute.