Paper 2-25

SAS® Software Formats: Going Beneath the Surface

Roger Staum, SAS Institute Inc., New York, NY

ABSTRACT

Through this tutorial, I hope to acquaint the experienced SAS programmer with advanced and not-so-well-known features of creating and using formats and informats. Most of these features are supported in Version 6 of the SAS® System, although a few are new to Version 8[1]. They apply to base SAS® software on all supported platforms. Although I emphasize concepts, I also include syntax, examples, and applications.

INTRODUCTION

Since this is an advanced tutorial, I assume that you already know how to[2]:

- create temporary and permanent value formats using the FORMAT procedure and the special keywords LOW, HIGH, and OTHER
- use FORMAT, INFORMAT, and ATTRIB statements, and PUT and INPUT functions
- maintain user-defined formats by modifying source code or via control data sets
- apply formats and informats to tasks such as reading external data, decoding, validation, collapsing data, type conversion, and table lookup.

This tutorial will cover the following topics:

- mapping concepts
- some useful system formats and informats
- creating value formats and informats
- referencing formats and informats

MAPPING CONCEPTS

A *mapping* from one set to another is a familiar concept. In mathematics, a *function* is a type of mapping. Its *domain* is defined as the set of all valid "input" values to be mapped, and its *range* is the set of all valid "output" values[3]. For example, if the domain of the function $\mathbf{f(x) = x^2}$ is defined as all real numbers, then its range is all real numbers ≥ 0. A function is constrained so that any value in its domain can be mapped to one and only one value in its range.

A mathematical function is a convenient model for thinking about formats and informats. They map individual values or intervals in a domain into individual values in a range. Instead of applying a simple formula such as $\mathbf{f(x) = x^2}$, however, they may apply various, sometimes complex transformations to different values.

Look at the following examples:

1) The system format DATE9. has a domain of all real numbers in the interval [-138061,2936547] and a corresponding range of dates in the time interval [01JAN1582,31DEC9999], expressed as character values. Fractional values in the domain are rounded down to the closest integer, creating a one-to-one correspondence between integers and dates in these intervals.
2) The system informat 2. has a domain of all 1- or 2-character strings consisting of the digits 0-9, as well as an optional decimal point and leading **+** or **-** signs. Its range consists of the numeric values which correspond to these strings.
3) The user-defined format $SEX, defined as:

```
VALUE $sex
        'M' = 'Male'
        'F' = 'Female'
        OTHER = 'Invalid';
```

has a domain of all possible character values up to a maximum length which is version-specific. Its range consists of the three character values **Male**, **Female**, and **Invalid**.

Throughout this tutorial, I will refer to the domain and range of a format or informat, rather than the terms *value-range-sets* and *labels* found in SAS documentation.

Formats and informats are categorized by type: character or numeric. Originally, an informat was a tool for reading external text or data into SAS, and a format was a tool for writing SAS data as external text or data. Since SAS data may be either numeric or character, we can summarize the domain and range restrictions of early releases of the SAS System as follows:

Formats
Character: SAS character value ➨ external text string
Numeric: SAS numeric value ➨ external text string

Informats
Character: external text string ➨ SAS character value
Numeric: external text string ➨ SAS numeric value

With the development of the INPUT and PUT functions, formats and informats were also able to map data that were entirely internal to SAS. Since SAS treats external text and data as character strings, we can think of formats and informats as defined over more general domains and ranges:

Formats
Character: character ➨ character
Numeric: numeric ➨ character

Informats
Character: character ➨ character
Numeric: character ➨ numeric

Notice that:

- it is the domain which determines whether a format is character or numeric, but it is the range which determines the type of an informat
- since the possible domain and range of character formats and informats are the same, they can sometimes be used interchangeably.

Later enhancements to formats and informats, mentioned below, modify further the domain and range patterns.

SOME USEFUL SYSTEM FORMATS AND INFORMATS

It is not a goal of this tutorial to discuss all the available formats and informats built into the SAS System. I would like to point out, however, a few which are not well known, but can be very useful.

$UPCASE*w*.

The informat will read, and the format will write, ***w*** bytes, changing lower-case letters to upper case. In many situations, this allows you to avoid calling the UPCASE function.

YYMMN*w*.

Suppose you need to read a character string like '9708' into a SAS date, assuming the 1st of the month. A traditional solution is:

```
DATA _NULL_;
        INPUT year 2.
              month 2.;
        date = MDY(month,1,year);
        DATALINES;
9708
RUN;
```

In Version 8 of the SAS System, you can read the data more easily and efficiently using this informat:

```
DATA _NULL_;
        INPUT date YYMMN4.;
        DATALINES;
   9708
RUN;
```

The corresponding format reverses the process, writing a SAS date as year and month numbers.

INTERNATIONAL DATE AND DATETIME FORMATS

Version 8 of the SAS System introduced a set of international (primarily European) date and datetime formats, all of whose names start with the letters EURDF. They correspond to the traditional, English-language formats as follows:

English	Example	International
DATE.	01JAN60	EURDFDE.
DATETIME.	01JAN60:00:00:00	EURDFDT.
DDMMYY.	01/01/60	EURDFDD.
DOWNAME.	Friday	EURDFDWN.
MONNAME.	January	EURDFMN.
MONYY.	JAN60	EURDFMY.
WEEKDATX.	Friday, 1 January 1960	EURDFWKX.
WEEKDAY.	6	EURDFDN.
WORDDATX.	1 January 1960	EURDFWDX.

The exact value of an element in the range of one of these formats depends on the setting of the new DFLANG= system option. For example:

```
44   OPTIONS DFLANG='French';
45   DATA _NULL_;
46           ReferenceDate = 0;
47           PUT ReferenceDate= EURDFWKX.;
48   RUN;
ReferenceDate=Vendredi 1er janvier 1960

50   OPTIONS DFLANG='German';
51   DATA _NULL_;
52           ReferenceDate = 0;
53           PUT ReferenceDate= EURDFWKX.;
54   RUN;
ReferenceDate=Freitag, 1. Januar 1960
```

INFORMAT ALIASES

For compatibility with other languages, or just to avoid unnecessary syntax errors, there are alternate versions (aliases) for some popular system informats:

Primary informat	Alias
COMMAw.d	**DOLLARw.d**
COMMAXw.d	**DOLLARXw.d**
w.d	**BESTw.d,, Dw.d, Fw.d, Ew.d**
$w.	**$Fw.**

CREATING FORMATS AND INFORMATS

This section discusses special features you can use, or important details to remember, when creating formats and informats with PROC FORMAT.

INFORMATS

Many SAS programmers know how to define their own formats, but are unaware that they can define their own informats, or how user-defined informats can be helpful.

The syntax is straight-forward. Instead of a VALUE statement, you write an INVALUE statement which contains mappings from a domain to a range.

When the SAS System first supported user-defined informats, domains were restricted to character values. If you listed numbers in the domain without quote marks, PROC FORMAT treated them as character (implied quotes). In Version 6.07 of the SAS System, however, numeric informats were enhanced to permit numeric domain values. For example:

```
PROC FORMAT;
        INVALUE numbers
                1-10 = 1
                11-20 = 2
                OTHER = 9;
RUN;

DATA _NULL_;
        INPUT value numbers.;
        PUT value=;
        DATALINES;
8
15
RUN;
```

```
value=1
value=2
```

It is even possible to define a domain for a numeric informat which contains both numeric and character values. For example:

```
PROC FORMAT;
        INVALUE mixed
                1-10 = 1
                11-20 = 2
                'XYZ' = 9
                OTHER = 999;
RUN;

DATA _NULL_;
        INPUT value mixed.;
        PUT value=;
        DATALINES;
8
100
XYZ
ABC
RUN;
```

```
value=1
value=999
value=9
value=999
```

This enhancement requires us to extend once more the permissible domains and ranges for formats and informats:

Formats
Character: character ➨ character
Numeric: numeric ➨ character

Informats
Character: character ➨ character
Numeric: character and/or numeric ➨ numeric

There are two special keywords which can be used in the range of an informat: _SAME_ and _ERROR_. _SAME_ indicates that a value in the domain is to be mapped into the same value in the range. _ERROR_ indicates that a value or set of values should be excluded from the domain. In practical terms, this means that the SAS System will perform its usual invalid data processing.

For example:

```
PROC FORMAT;
        INVALUE tryit
                1-10 = _SAME_
                OTHER = _ERROR_;
RUN;
```

```
DATA _NULL_;
        INPUT value tryit.;
        PUT value=;
        DATALINES;
5
15
RUN;
```

```
value=5
NOTE: Invalid data for value in line 27 1-12.
value=.
RULE:----+----1----+----2----+----3----+-
27   15
value=. _ERROR_=1 _N_=2
```

There are many applications for user-defined informats. The following example combines two applications: numeric encoding and data validation. Suppose a student's grades are reported as letters of the alphabet in an external file, and you need to calculate the student's GPA:

```
PROC FORMAT;
        INVALUE grades
                'A' = 4
                'B' = 3
                'C' = 2
                'D' = 1
                'F' = 0
                OTHER = _ERROR_;
RUN;

DATA _NULL_;
        INPUT name $
              (grade1-grade5) (: grades.);
        gpa = MEAN(OF grade1-grade5);
        PUT name=
            gpa=;
        DATALINES;
Joe B A H D C
RUN;
```

```
NOTE: Invalid data for grade3 in line 84 9-9.
name=Joe gpa=2.5
RULE:----+----1----+----2----+----3----+----4---
84   Joe B A H D C
name=Joe grade1=3 grade2=4 grade3=. grade4=1
grade5=2 gpa=2.5 _ERROR_=1 _N_=1
```

Two INVALUE statement options, JUST and UPCASE, can also be useful. JUST automatically left-justifies, and UPCASE automatically upcases, all input values before they are compared to the informat domain. UPCASE can be illustrated in the above example:

```
PROC FORMAT;
        INVALUE grades (UPCASE)
                'A' = 4
                'B' = 3
                'C' = 2
                'D' = 1
                'F' = 0
                OTHER = _ERROR_;
RUN;

DATA _NULL_;
        INPUT name $
              (grade1-grade5) (: grades.);
        gpa = MEAN(OF grade1-grade5);
        PUT name=
            gpa=;
        DATALINES;
Joe b a h d c
RUN;
```

The log messages and the value of GPA are identical to those of the previous example.

NAMES

Format names are restricted to 8 characters, even under Version 8 of the SAS System. Informat names are restricted to 7 characters, because internally the SAS system inserts a **@** as the 1st character of the name.[4] The leading **$**, needed to distinguish numeric and character formats and informats, counts in the above limits. The SAS System automatically truncates longer names, printing a log note such as:

```
NOTE: The format name '$ABCDEFGH' exceeds 8
characters. Only the first 8 characters will
be used.
```

OVERLAPPING

At one time or another, most SAS programmers have submitted code such as:

```
VALUE age
              LOW - 12 = 'Child'
              13 - 19 = 'Teenager'
              18 - 64 = 'Adult'
              65 - HIGH = 'Senior';
```

and seen a log message such as:

```
ERROR: These two ranges overlap: 13-19 and 18-64
(fuzz=1E-12).
NOTE: The previous statement has been deleted.
```

The word *ranges* in the error message refers to values or intervals on the left side of the **=** sign, not the range of the mapping. Translated into the terminology of this paper, the error message is saying that the format is attempting to map some elements in the domain into two or more elements in the range. For instance, 18.5 is mapped into both *teenager* and *adult*. This violates the definition of a mathematical function, and is usually rejected by PROC FORMAT, for both formats and informats.

There are exceptions to this rule. Two intervals which overlap at one point are acceptable to PROC FORMAT, with the point resolved in favor of the lower interval. For example, if I define the AGE format as:

```
VALUE age
              LOW - 13 = 'Child'
              13 - 19 = 'Teenager'
              20 - 64 = 'Adult'
              65 - HIGH = 'Senior';
```

then a value of **13** will be mapped to **Child**.

If this is not how you want the value resolved, or in order to remove any confusion, you should use **<** to indicate an excluded endpoint (open interval):

```
VALUE age
              LOW - <13 = 'Child'
              13 - <20 = 'Teenager'
              20 - <65 = 'Adult'
              65 - HIGH = 'Senior';
```

This version of the format will map **13** to **Teenager**, but **12.99** to **Child**.

Version 8 of the SAS System has added a VALUE statement option named MULTILABEL, which does allow domain elements to be mapped to multiple range elements. PROC FORMAT will accept the following code:

```
VALUE age (MULTILABEL)
              LOW - 12 = 'Child'
              13 - 19 = 'Teenager'
              18 - 64 = 'Adult'
              65 - HIGH = 'Senior';
```

Ordinary references to this format will resolve in favor of the lower interval. (A DATA step PUT statement would map **18.5** to **Teenager**, for example.)

There are applications in which data values belong to multiple categories, i.e. they are not strictly hierarchical. A few Version 8 summary procedures have been enhanced so that they can use multilabel formats to support such applications.

Supose we have a character format which maps musicians into two categories: **Classical** or **Popular**. Although most musicians fall into only one category, some belong to both:

```
PROC FORMAT;
     VALUE $music (MULTILABEL)
            'Barry Manilow' = 'Popular'
            'Artur Rubinstein' = 'Classical'
            'Leonard Bernstein' = 'Classical'
            'Leonard Bernstein' = 'Popular'
            'Madonna' = 'Popular'
            'Kurt Masur' = 'Classical';
RUN;
```

If we create a data set with 5 observations, containing the above 5 names, we can request the TABULATE procedure to summarize the data. Notice the MLF option on the CLASS statement:

```
DATA a;
        INPUT name $20.;
        DATALINES;
Barry Manilow
Artur Rubinstein
Leonard Bernstein
Madonna
Kurt Masur
RUN;

PROC TABULATE
     FORMAT=9.
     DATA=a;
        FORMAT name $music.;
        CLASS name / MLF;
        TABLE (name ALL) * N;
RUN;
```

name		
Classical	Popular	All
N	N	N
3	3	5

Leonard Bernstein is counted in both categories, but only once in the ALL column. This emphasizes the important point that ALL in PROC TABULATE does not simply add numbers across rows or columns, but is an independent summarization of the data with categories collapsed.

Use multilabel formatting with caution! You will probably tire of people coming into your office to tell you that 3 + 3 = 6. Head them off at the pass by adding a footnote explaining that some observations fall into more than one category, so numbers may not add up across rows or columns.

The MEANS and SUMMARY procedures also support the MLF option in the CLASS statement. Additional procedures may support it in future releases.

CHARACTER INTERVALS

Character-valued intervals sometimes confuse programmers. The interval **A-C**, for instance, matches any character value, such as **BX**, starting with A, B, or C.

FUZZY DOMAINS

You may want a value to be mapped if it is "close enough" to an element in the domain of a numeric format or informat. The FUZZ option on the VALUE or INVALUE statement will allow you to do this:

```
PROC FORMAT;
        VALUE close (FUZZ=0.1)
                1 = 'One';
RUN;

DATA _NULL_;
     INPUT number;
     PUT number= close.;
     DATALINES;
0.8
0.9
1.0
1.1
1.2
RUN;
```

Values are mapped to **One** if they fall within the closed interval [0.9,1.1]:

```
number=0.8
number=One
number=One
number=One
number=1.2
number=6
```

The default fuzz value is 1E-12.

NESTING

There are situations in which no one format or informat will suffice or is desirable as a mapping across an entire domain. In a statistical report, for instance, you may want to display p-values with more decimal places as the values approach 0.[5] Starting with Version 6.07, the SAS System provides a solution with nested formats and informats, in which an element of the range can point to another format or informat:

```
 PROC FORMAT;
     VALUE pvalfmt
        0 - 0.001     = [6.4]
        0.001< - 0.01 = [5.3]
        0.01< - 1     = [4.2]
        OTHER         = 'Invalid';
RUN;

DATA _NULL_;
        INPUT number;
        PUT number= pvalfmt.;
        DATALINES;
0.000006
0.00006
0.0006
0.006
0.06
0.6
6
RUN;
```

The output from the PUT statement is:

```
number=0.0000
number=0.0001
number=0.0006
number=0.006
number=0.06
number=0.60
number=Invalid
```

The "secondary" formats and informats can be user-defined, as well as system-supplied. For example:[6]

```
PROC FORMAT;
        VALUE fruit
                1 = 'banana'
                2 = 'apple'
                3 = 'pear';
        VALUE veg
                51 = 'broccoli'
                52 = 'lima beans'
                53 = 'peas';
```

```
        VALUE dairy
                101 = 'skim milk'
                102 = 'cheese'
                103 = 'yogurt';
         VALUE meat
                151 = 'ground beef'
                152 = 'pork chops'
                153 = 'sirloin';
         VALUE foodcats
                1-50 = [fruit12.]
                51-100 = [veg12.]
                101-150 = [dairy12.]
                151-200 = [meat12.];
RUN;

DATA _NULL_;
        INPUT foodcode @@;
        PUT foodcode= foodcats.;
        DATALINES;
1 53 103 152
RUN;
```

```
foodcode=banana
foodcode=peas
foodcode=yogurt
foodcode=pork chops
```

The advantage of this nested, modular approach is that the secondary formats (e.g. **dairy**.) can be maintained and invoked separately from the primary format (**cats**.).

Recursion is forbidden using this technique (a format or informat cannot point back to itself). Formats and informats can be nested more than one level, but performance deteriorates.

OVERLOADING THE DOMAIN OR RANGE

An oft-cited disadvantage of using formats and informats for table lookup is that each lookup action can return only one value. In contrast, a data set used as a lookup table can return many values, since it may contain many variables.

You can overcome this disadvantage in either of two ways:

- by creating several formats or informats
- by *overloading*[7] the range of a single format or informat. This means that you define range elements which contain several pieces of information, separated by a unique, consistent delimiter. The user of the format or informat extracts a desired piece of information using the PUT or INPUT and SCAN functions.

Suppose, for example, you define a format which maps state abbreviations into the year in which the state was admitted to the Union, as well as its official flower, bird, and tree:

```
PROC FORMAT;
   VALUE $states
    'NY' = '1788/Rose/Bluebird/Sugar maple'
    'IN' = '1816/Peony/Cardinal/Tulip poplar'
    'NC' = '1789/Dogwood/Cardinal/Pine';
RUN;
```

To determine the state bird of Indiana, you would submit code such as:

```
123  DATA _NULL_;
124       LENGTH bird $10;
125       bird = SCAN(PUT('IN',$states.),3,'/');
126       PUT bird=;
127  RUN;
bird=Cardinal
```

Overloading a format or informat domain is analogous to defining a multivariate function such as **f(x,y,z)** in mathematics. It allows you to do a lookup based on several pieces of information. For instance, in a clinical trials application, you might create an informat which maps subject, period, and visit numbers into visit dates:

```
PROC FORMAT;
     INVALUE vstdates
                '1010101' = '15FEB2000'D
                '1010102' = '22FEB2000'D
                '1010201' = '15MAR2000'D;
RUN;

DATA a;
     INPUT subject $3. +1
           period  $2. +1
           visit   $2.;
     VisitString=subject||period||visit;
     VisitDate=INPUT(VisitString,vstdates.);
     PUT VisitDate= MMDDYY10.;
     DATALINES;
101 01 02
RUN;
```

```
VisitDate=02/22/2000
```

Whether you are overloading the domain or the range, you are likely to create the format or informat using control data sets.

REFERENCING FORMATS AND INFORMATS

This section deals with referencing system-supplied or user-defined formats and informats.

DEFAULTS

The SAS System assigns default formats and informats to variables in any of the following situations:

- You read data from an external file using column- or list-style input.
- You create a variable and request that its value be written to an external file or to a report, without assigning a format, or canceling an existing format.
- You force implicit type conversion.
- You have set the system option NOFMTERR, and the assigned format cannot be found or loaded.

In any of these situations, the SAS System uses the appropriate default tool:

Numeric format: BEST12.

Character format: $*w*., where *w* is the length of the character variable.

Numeric informat: *w*.

Character informat: $*w*.

where *w* is the number of bytes in the column interval (column-style input), the number of bytes between delimiters (list-style input), or the length of the character variable (implicit character-to-numeric conversion).

USING SHORT FORMAT WIDTHS

The width of a user-defined format is equal to the length of the longest character string in its range. You can truncate the range values by referencing the format with a shorter width.

For instance, suppose you have defined a format named MONTHS which maps the integers 1 through 12 to the names of the months. Internally, the SAS system refers to this format as MONTHS9., since the longest month name has 9 letters. This format looks good when used in a PRINT procedure report:

```
Month

January
January
January
January
January
January
February
February
```

```
February
February
February
```

but the range values are inconveniently wide in a PROC TABULATE report:

January	February	March	April	May	
6	9	2	8	1	

You may be tempted to create another format, using only the first three letters of each month. This is unnecessary, however. Simply refer to your existing format as MONTHS3.:

```
PROC TABULATE
     FORMAT=3.
     DATA=a;
        CLASS Month;
        TABLE Month=' '*N=' ';
        FORMAT Month months3.;
RUN;
```

Jan	Feb	Mar	Apr	May	Jun
6	9	2	8	1	8

COMPOSITION

In mathematics, you can define a new function **h** by *composing* functions **f** and **g** in a particular sequence, e.g. **h = f ° g**. In order for this composition to be valid, the range of **g** must be a subset of the domain of **f**. For example, suppose **g** maps all real numbers into all non-negative real numbers via the rule $\mathbf{g(x) = x^2}$, and **f** maps all real numbers into all real numbers via the **rule f(x) = 2x**. Since the range of **g** is a subset of the domain of **f**, the composition **h = f ° g** is valid, and $\mathbf{h(x) = f(g(x)) = 2x^2}$.

Similarly, two formats or informats may be referenced in sequence, using the PUT and/or INPUT functions, if the range of the first is a subset of the domain of the second. For instance, suppose you have a format which maps salespeople to regions, and a format which maps regions to managers. Then you can look up any salesperson's manager by nesting calls to the PUT function:

```
PROC FORMAT;
        VALUE $spreg
                'Smith' = 'NE'
                'Jones' = 'NE'
                'Brown' = 'NW'
                'Baker' = 'SE';
        VALUE $regmgr
                'NE' = 'Witherspoon'
                'NW' = 'McGillicuddy'
                'SE' = 'Stringfellow'
                'SW' = 'Rumplestiltskin';
RUN;

DATA _NULL_;
     SalesPerson = 'Jones';
      Manager=PUT(PUT(SalesPerson,$spreg.),
            $regmgr.);
     PUT Manager=;
RUN;
```

```
Manager=Witherspoon
```

As another example, suppose you have Mozart's birthday stored as a COBOL-style numeric value, and would like to create a new variable in which this information is stored as a SAS date. You can convert the numeric value to character using the 8. format, and then read the character value as a SAS date using the YYMMDD. informat:

```
DATA _NULL_;
 COBOLStyleDate = 17560127;
 SASdate=INPUT(PUT(COBOLStyleDate,8.),YYMMDD8.);
 PUT SASdate = DATE9.;
RUN;
```

```
SASdate=27JAN1756
```

SUPPRESSING INVALID DATA MESSAGES

If an informat referenced in an INPUT statement attempts to read a data value which is not an element in that informat's domain, the SAS System will take the following actions:

- write an "invalid data" message in the log
- show the contents of the input buffer and program data vector in the log
- set _ERROR_ to 1
- set the receiving variable to missing.

If an informat referenced in an INPUT function attempts to read a data value which is not an element in that informat's domain, the SAS System will take the following actions:

- write an "invalid argument to function" message in the log
- show the contents of the program data vector in the log
- set _ERROR_ to 1.
- set the receiving variable to missing.

You might want to suppress some of these actions:

- because the data are under the control of another person, and you do not want to frighten that person unnecessarily, should he/she happen to read the log
- because the invalid data do not really constitute a problem, and you do not want to be distracted by the messages in the log.

You can do this by inserting one or two question marks in front of the informat reference. **?** suppresses the invalid data/argument message; **??** suppresses everything other than the setting of the receiving variable to missing. For example:

```
650  DATA _NULL_;
651          INPUT date ?? DATE9.;
652          PUT _INFILE_ /
653               date;
654          DATALINES;

30FEB2000
.
NOTE: DATA statement used:
      real time           0.06 seconds
      cpu time            0.01 seconds
```

RUN-TIME BINDING

In most situations, you know enough about your data to determine the correct format or informat to use. Occasionally, however, you cannot choose the right format or informat unless you examine the data first.

The issue underlying this problem is called *binding*, which refers to the time at which a meaning or value is assigned (or bound) to a symbolic. When most formats and informats are being used in INPUT or PUT statements or functions, they must be bound at *compile-time*. This means that the code which is received by the DATA step compiler or PROC parser must provide the complete name and width of any format or informat which you reference. But you need *run-time* binding if the details of the format or informat cannot be determined until the step executes and data are read.

If you are willing to read the data twice, you can write the program so that you decide on the right tool on the first pass, put its name in a macro variable, and reference that macro variable in the second pass. This approach is inefficient, sometimes impractical, and often unnecessary.

There are tools which directly implement run-time binding of formats and informats. If you would like to use the $*w*. format or informat, but the value of *w* must vary based on the data, then $VARYING. often solves the problem. For more complex situations, when even less may be known at compile time about the appropriate format or informat, the PUTC, PUTN, INPUTC, or INPUTN functions, added in Version 6.07, may provide the right solution. These are generalizations of the PUT and INPUT functions, in which you can construct the format or informat name from an expression at run time.

For example[8], suppose there are three standard responses – **positive**, **negative**, or **neutral** - to three different questions, but the true meanings of these responses vary by the question. One solution is to:

1) map the question number into an informat name using a format and the PUT function, then
2) map the external response to the more specific response by choosing the correct informat with the INPUTC function.

```
PROC FORMAT;
     VALUE qfmt
             1='$qa'
             2='$qb'
             3='$qc';
     INVALUE $qa
             'positive'='agree'
             'negative'='disagree'
             'neutral'='notsure';
     INVALUE $qb
             'positive'='accept'
             'negative'='reject'
             'neutral'='possible';
     INVALUE $qc
             'positive'='pass'
             'negative'='fail'
             'neutral'='retest';
RUN;

DATA answers;
     INPUT question response $;
     respinformat = PUT(question,qfmt.);
     word = INPUTC(response,respinformat);
     DATALINES;
1 positive
1 negative
1 neutral
2 positive
2 negative
2 neutral
3 positive
3 negative
3 neutral
RUN;

PROC PRINT
     DATA=answers
     NOOBS;
RUN;
```

question	response	respinformat	word
1	positive	$qa	agree
1	negative	$qa	disagree
1	neutral	$qa	notsure
2	positive	$qb	accept
2	negative	$qb	reject
2	neutral	$qb	possible
3	positive	$qc	pass
3	negative	$qc	fail
3	neutral	$qc	retest

MACRO ENVIRONMENT

Until recently, formats and informats were not directly available in the macro environment. If, for instance, you wanted to put today's date in a title or footnote, you needed to add a DATA step in order to call the TODAY function, convert the resulting SAS date value with an appropriate date format, and pass it to the macro symbol table using CALL SYMPUT.

Version 6.11 of the SAS System introduced a more direct solution - which has no need for a DATA step - with a macro function called SYSFUNC. This function makes almost all DATA step functions available to the macro environment. To use SYSFUNC, you need to remember that:

- you cannot use the PUT and INPUT functions. You must use PUTN, PUTC, INPUTN, or INPUTC
- you cannot nest the non-macro function calls, but you can nest calls to SYSFUNC
- character arguments which need quote marks in a DATA step generally do not need them in the macro environment
- you may need to use the QSYSFUNC function to mask the meaning of special characters from the macro processor.

Using SYSFUNC, the solution to the above problem becomes:

```
TITLE1 "Report produced
%sysfunc(PUTN(%sysfunc(TODAY()),MMDDYY10.))";
PROC PRINT DATA=a;
RUN;
```

```
Report produced 01/30/2000

Obs    name      sex

 1     Susan     Female
 2     Joe       Male
```

CONCLUSION

Formats and informats are powerful tools for transforming data values. It may help you understand their capabilities by comparing them to mathematical functions. I hope I have given you new ideas for enhancing your programs, using advanced features of creating and using formats and informats.

CONTACT INFORMATION

Your comments and questions are welcome. Contact the author at:

Roger Staum
SAS Institute Inc.
787 Seventh Ave., 47th Floor
New York, NY 10019
Email: Roger.Staum@sas.com

[1] When I mention a feature as being new to Version 8, it may actually have first appeared in Version 7.
[2] These are topics covered in SAS Institute courses.
[3] This is a simplified definition of the range. Strictly speaking, the set of all valid values produced by a function is its image, which may be a subset of the range. For purposes of this tutorial, however, I will use the simplified definition.
[4] The only place you need to include the @ is in SELECT or EXCLUDE statements in PROC FORMAT.
[5] This is a simplified version of an example sent to me by Chris Bond of the UK office of SAS Institute.
[6] This example is adapted from a tutorial by Rick Langston of the Cary, NC office of SAS Institute. It has been presented at several user group meetings.
[7] Marjorie Lampton of the Overland Park, KS office of SAS Institute suggested the term *overloading*.
[8] This example is adapted from SAS OnlineDoc®, Version 8.