**PharmaSUG 2015 – Paper BB09**

# Things Are Not Always What They Look Like: PROC FORMAT in Action

Peter Eberhardt, Fernwood Consulting Group Inc, Toronto, Canada
Lucheng Shao, Ivantis Inc, Irvine, CA, U.S

## ABSTRACT

When we first learn SAS®, we quickly learn the value of built-in formats to convert our underlying data into visual representations fit for human **C**onsumption. For many SAS programmers that is as far as their understanding of formats goes. In this paper we will show creating and using formats and informats created with PROC FORMAT. We will show how you can use these artifacts to do more than just change the appearance of a variable.

## INTRODUCTION

The fundamental purpose of the data in our datasets is for analysis, letting us move from having collections of disparate values to having a cohesive set of data from which we can make decisions to improve our business. However, at almost every step along the way there is the need for a person to read the data – to see a visual representation of these data. In the SAS, one of the most obvious needs for this visual representation is with SAS date values. Even the most diehard SAS geek will struggle with the meaning of 20225, but will understand 2015-05-17, or May 17, 2015, the visual representation of the SAS date value 20225. In this paper we will look at creating and using SAS formats , and by extension informats, through the use of PROC FORMAT.

## DEFINE YOUR OWN FORMATS AND INFORMATS

Although SAS provides a large number of build-in formats and informats, often times SAS programmers would still need to define their own formats and informats to meet specific data formatting requirements. The general form of PROC FORMAT is:

```
PROC FORMAT;
VALUE FORMAT_NAME
     DATA_VALUE1 = FORMAT_LABEL1
     DATA_VALUE2 = FORMAT_LABEL2
     DATA_VALUE3 = FORMAT_LABEL3
     ⊤
     ;
RUN;
```

In clinical trials a common example could be:

```
PROC FORMAT;
 VALUE VISIT_NO
       -2 = 'SCREENING'
       -1 = 'BASELINE'
        0 = 'SURGERY'
        1 = '1 DAY'
        2 = '1 WEEK'
        3 = '1 MONTH'
        4 = '3 MONTH'
        5 = '6 MONTH'
        ;
RUN;
```

We created a numeric format called VISIT_NO. By applying this VISITNO format on the visit_no variable, numeric visit numbers are converted to character labels showing the name of the visits. This makes it easy for data review.

Another example could be:

```
PROC FORMAT;
  VALUE $ VISIT_NAME
  'SCREENING' = -2
  ‘BASELINE’  = -1
  'SURGERY'   =  0
  '1 DAY'     =  1
  '1 WEEK'    =  2
  '1 MONTH'   =  3
  '3 MONTH'   =  4
  '6 MONTH'   =  5
  ;
RUN;
```

This time we created a character format called VISIT_NAME. The $ sign indicates the VISIT_NAME format is character rather than numeric. By applying this VISIT_NAME format on the visit_name variable, character visit names are converted to numeric numbers. This makes it easy for further data sorting and calculation.

FORMAT is used when the variable is already in the SAS database and we want to change the printed appearance of this variable; while INFORMAT is used to read in a variable from external data source and store it in a particular format in the generated SAS dataset.

```

PROC FORMAT;
 INVALUE VISIT_NO
       -2 = 'SCREENING'
       -1 = 'BASELINE'
        0 = 'SURGERY'
        1 = '1 DAY'
        2 = '1 WEEK'
        3 = '1 MONTH'
        4 = '3 MONTH'
        5 = '6 MONTH'
        ;
RUN;
```

We created a character informat called VISIT_NO. It is applied when reading in raw dataset VISIT.

```
DATA VISIT;
INPUT VISIT_NO $VISIT_NO.;
DATALINES;
1
2
0
41
;
```

Note that the type of the format (numeric or character) is defined by the left-hand side of the equal sign in PROC FORMAT while the type of the informat is defined by the right-hand side of the equal sign. This is because the created format will be applied on a variable that is already in the SAS dataset and the type of the left-hand side is consistent with the type of this variable. However, an informat is used to read in a variable from the raw data and at the same time define the type of that variable in the SAS dataset. It is the right-hand side of the equal sign in PROC FORMAT that defines the value and the type of the variable that will be stored as.

Using PROC PRINT, the output of VISIT dataset looks like:

| Obs | VISIT_NO |
|---|---|
| 1 | 1 DAY |
| 2 | 1 WEEK |
| 3 | SURGERY |
| 4 | 41 |

The value of 41 was kept as is in the output because there is no related label value that is created for this 41 in the VISIT_NO informat.

In clinical trials, for some reason (eg. for the purpose of adverse events follow up), we could have unscheduled visits. Those unscheduled visits will also have their related visit numbers. Any other value in visit_no variable will be assumed as data entry mistake. In the following example informat VISIT_NO_DEBUG with _ERROR_ option can help us catch such mistakes. Note that the _ERROR_ option only applies with informats (not formats).

```
PROC FORMAT;
 INVALUE $VISIT_NO_DEBUG
  -2 = 'SCREENING'
  -1 = 'BASELINE'
   0 = 'SURGERY'
   1 = '1 DAY'
   2 = '1 WEEK'
   3 = '1 MONTH'
   4 = '3 MONTH'
   5 = '6 MONTH'
      11, 12, 13 = 'UNSCHEDULED'
   OTHER = _ERROR_
   ;
RUN;

DATA VISIT_BUG;
INPUT VISIT_NO $VISIT_NO_DEBUG.;
DATALINES;
1
2
0
11
41
;
```

Using PROC PRINT, the output of VISIT_BUG dataset looks like:

| Obs | VISIT_NO |
|---|---|
| 1 | 1 DAY |
| 2 | 1 WEEK |
| 3 | SURGERY |
| 4 | UNSCHEDULED |
| 5 | |

Since the number 41 in the raw data doesn't have any related label value defined in the informat VISIT_NO_DEBUG, the corresponding place is left as blank in the output. At the same time, an ERROR message printed out in the log file.

```
NOTE: Invalid data for VISIT_NO in line 596 1-11.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8---
-+--
596        41
VISIT_NO=  _ERROR_=1 _N_=5
```

## CREATE A PICTURE FORMAT WITH PRE-DEFINED TEMPLATE

SAS beginners may not be familiar with the picture format which can be a very powerful tool to help you set up a pre-defined template for your format. In this pre-defined format, you could:

- Set up the printed-out digits and decimals of numbers in the output by using '9' and '0' to form the template
- Choose whether to have leading zeros
- Choose whether to truncate extra digits or rounding off when the number of decimals of a number exceeds the template-allowed number of decimals
- Append text to numbers

Use the following dataset as an example:

```
DATA MEDS_DOSE;
INPUT SUBJECT MEDS $ DOSE;
DATALINES;
001 MED_A  45.74
002 MED_B  89.3
003 MED_C   7.4
004 MED_D  120.527
;
```

```
PROC FORMAT;
  PICTURE DOSE_A
          LOW - HIGH = '99.99'
           ;
RUN;

PROC FORMAT;
  PICTURE DOSE_B
          LOW - HIGH = '09.90'
           ;
RUN;
```

The output of the DOSE column in DOSE_A format and DOSE_B format are summarized below:

| SUBJECT | MEDS | DOSE without format | DOSE in DOSE_A format '99.99' | DOSE in DOSE_B format '09.90' |
|---|---|---|---|---|
| 1 | MED_A | 45.74 | 45.74 | 45.74 |
| 2 | MED_B | 89.3 | 89.30 | 89.30 ① |
| 3 | MED_C | 7.4 | 07.40 | 7.40 ② |
| 4 | MED_D | 120.527 | 20.52 | 20.52 ③ |

Both '9' and '0' are used as place holder. When originally the digit is not in the number, both '9' and '0' will force adding trailing zeros (see ①); '9' will also forces adding leading zeros while '0' will not (see ②).

Note that extra leading and trailing digits are truncated in both cases (see ③), and there is no warning message reported in log file. This warns us that we have to be very careful when setting up the number of digits in the template so that no digits will be lost without awareness.

With the extra trailing digits we could choose to do rounding off rather than truncating by simply adding the ROUND option in the picture format.

```
PROC FORMAT;
  PICTURE DOSE_A (ROUND)
          LOW - HIGH = '99.99'
      ;
RUN;

PROC FORMAT;
  PICTURE DOSE_B (ROUND)
          LOW - HIGH = '09.90'
      ;
RUN;
```

Now the number 120.527 turned out to be 20.53 in both cases.

You could also create a template having explaining text append to numbers for data review convenience. For example:

```
PROC FORMAT;
  PICTURE DOSE_C
          LOW - < 30 = 'LOW DOSE'
      30 - 100 = '99.99 - NORMAL DOSE'
      100 - HIGH = 'HIGH DOSE'
          ;
RUN;
```

By applying DOSE_C format on DOSE variable, the output of MEDS_DOSE becomes:

| Obs | SUBJECT | MEDS | DOSE |
|---|---|---|---|
| 1 | 1 | MED_A | 45.74 - NORMAL DOSE |
| 2 | 2 | MED_B | 89.30 - NORMAL DOSE |
| 3 | 3 | MED_C | LOW DOSE |
| 4 | 4 | MED_D | HIGH DOSE |

## FORMAT YOUR MISSING DATA TO SHOW DIFFERENT MISSING REASONS

In clinical studies missing data issue is very common. A missing data point can be a result of several different reasons (eg., subject lost to follow up, visit not done, etc.).

In SAS a missing character is normally represented by a blank (‘ ‘) and a missing numeric value is represented by a period (‘.’). If we simply use a period (‘.’) to represent all kinds of missing numeric values, the information about missing reasons will be lost. If we use text to record missing reasons, some reason could be complicated and thus make the dataset lengthy. Creating a format using special characters to represent different missing reasons and apply the format on missing data can help remember the missing reason and at the same time keep the SAS dataset as succinct as possible.

These special characters include ‘.A’ to ‘.Z’ and ‘._’. When blended with regular numeric values, the sort order is ‘._’ <‘.’ (regular missing numeric values) < ‘.A’ < ‘.Z’ < negative values < 0 < positive values. In the following an example will be shown to walk you through how to create such a format and how we could benefit from applying such a format on missing data. Suppose we have a raw dataset in Excel from an ophthalmic device study:

| Subject | OE | Trt_group | VisitDt | Visit_Name | DIOP_OD | DIOP_OS |
|---|---|---|---|---|---|---|
| 111 | OD | (Not assigned) | 5/19/2011 | Screening | 19.6 | 21.5 |
| 112 | OD | (Not assigned) | 5/25/2011 | Screening | 26.7 | 24.9 |
| 113 | OS | (Not assigned) | 6/3/2011 | Screening | 17.2 | 18.8 |
| 114 | OS | (Not assigned) | 7/3/2011 | Screening | 15.4 | 14.7 |
| 111 | OD | (Not assigned) | 6/29/2011 | Baseline | 21.3 | (Non Study Eye) |
| 112 | OD | (Not assigned) | 6/30/2011 | Baseline | 34.6 | (Non Study Eye) |
| 113 | OS | (Not assigned) | 7/7/2011 | Baseline | (Non Study Eye) | (Not Washed Out) |
| 114 | OS | (Exit) | (Exit) | Baseline | (Exit) | (Exit) |
| 111 | OD | 1 | 7/1/2011 | Postop_1 Day | 18.2 | (Non Study Eye) |
| 112 | OD | 2 | 7/2/2011 | Postop_1 Day | 24.3 | (Non Study Eye) |
| 113 | OS | 2 | 7/9/2011 | Postop_1 Day | (Non Study Eye) | 20.8 |
| 114 | OS | (Exit) | (Exit) | Postop_1 Day | (Exit) | (Exit) |
| 111 | OD | 1 | (Lost to follow up) | Postop_1 Month | (Lost to follow up) | (Non Study Eye) |
| 112 | OD | 2 | 7/28/2011 | Postop_1 Month | 21.8 | (Non Study Eye) |
| 113 | OS | 2 | 8/6/2011 | Postop_1 Month | (Non Study Eye) | 18.1 |
| 114 | OS | (Exit) | (Exit) | Postop_1 Month | (Exit) | (Exit) |
| 111 | OD | 1 | (Lost to follow up) | Postop_12 Month | (Lost to follow up) | (Non Study Eye) |
| 112 | OD | 2 | (Not Done) | Postop_12 Month | (Not Done) | (Non Study Eye) |
| 113 | OS | 2 | 6/20/2012 | Postop_12 Month | (Non Study Eye) | 17.9 |
| 114 | OS | (Exit) | (Exit) | Postop_12 Month | (Exit) | (Exit) |

Note: In this example, the text within parentheses () indicates the reason for missing data points.
Notations: OE = Study Eye; OD = Right Eye; OS = Left Eye; Trt_group = Treatment Group; VisitDt = Visit Date; Visit_Name= the name of the visit; DIOP_OD = DIOP value of the right eye; DIOP_OS = DIOP value of the left eye.

We create a format called MISSING using special characters and let each special character represent a certain missing reason. All the missing information in the raw dataset is substituted with special character that is related to the missing reason. Using PROC PRINT, we are expected to see all the special missing characters are printed out with their missing reason labels in the output. However, we got an error message in the log file.

```
PROC FORMAT;
VALUE MISSING .R = 'NOT RANDOMIZED'
              .E = 'EXIT'
              .L = 'LOST TO FOLLOW UP'
              .D = 'NOT DONE'
             .N = 'NOT AVAILABLE'
              .W = 'NOT WASHED OUT'
               ;
RUN;
```

**Modified Example 4 with all missing values represented by special characters.**

| Subject | OE | Trt_group | VisitDt | Visit_Name | DIOP_OD | DIOP_OS |
|---|---|---|---|---|---|---|
| 111 | OD | .R | 5/19/2011 | Screening | 19.6 | 21.5 |
| 112 | OD | .R | 5/25/2011 | Screening | 26.7 | 24.9 |
| 113 | OS | .R | 6/3/2011 | Screening | 17.2 | 18.8 |
| 114 | OS | .R | 7/3/2011 | Screening | 15.4 | 14.7 |
| 111 | OD | .R | 6/29/2011 | Baseline | 21.3 | .N |
| 112 | OD | .R | 6/30/2011 | Baseline | 34.6 | .N |
| 113 | OS | .R | 7/7/2011 | Baseline | .N | .W |
| 114 | OS | .E | .E | Baseline | .N | .E |
| 111 | OD | 1 | 7/1/2011 | Postop_1 Day | 18.2 | .N |
| 112 | OD | 2 | 7/2/2011 | Postop_1 Day | 24.3 | .N |
| 113 | OS | 2 | 7/9/2011 | Postop_1 Day | .N | 20.8 |
| 114 | OS | .E | .E | Postop_1 Day | .N | .E |
| 111 | OD | 1 | .L | Postop_1 Month | .L | .N |
| 112 | OD | 2 | 7/28/2011 | Postop_1 Month | 21.8 | .N |
| 113 | OS | 2 | 8/6/2011 | Postop_1 Month | .N | 18.1 |
| 114 | OS | .E | .E | Postop_1 Month | .N | .E |
| 111 | OD | 1 | .L | Postop_12 Month | .L | .N |
| 112 | OD | 2 | .D | Postop_12 Month | . | .N |
| 113 | OS | 2 | 6/20/2012 | Postop_12 Month | .N | 17.9 |
| 114 | OS | .E | .E | Postop_12 Month | .N | .E |

```
LOG:
20603  PROC PRINT DATA=EX;
20604  FORMAT TRT_GROUP VISITDT DIOP_OD DIOP_OS MISSING.;
ERROR: YOU ARE TRYING TO USE THE NUMERIC FORMAT MISSING WITH THE CHARACTER VARIABLE
VISITDT IN DATA SET WORK.EX.
20605  RUN;

NOTE: THE SAS SYSTEM STOPPED PROCESSING THIS STEP BECAUSE OF ERRORS.
NOTE: PROCEDURE PRINT USED (TOTAL PROCESS TIME):
REAL TIME           0.00 SECONDS
CPU TIME            0.00 SECONDS
```

The error went with the VISITDT variable. The rule for importing a date variable from an Excel file to SAS is that if the data variable does not contain any character, it will be imported as a numeric variable; otherwise, it will be treated as character. In the above example, the VISITDT variable includes special characters '.E', '.L' and '.D' and thus it is treated as character. To solve the issue, we need to create a pair of informat and format (both called MISSDATES). Use the informat MISSDATES to read in VISITDT and save it as a numeric variable by applying INPUT function, and then apply format MISSDATES to get us the right output.

```
proc format;
invalue missDates
        .E      = .E
        .L      = .L
        .D      = .D
        other   = [ANYDATE10.]
        ;

value missDates
         .E      = 'Exit'
         .L      = 'Lost to follow up'
         .D      = 'Not Done'
         other   = [yymmdd10.]
         ;
run;


data ex_2(drop=dt);
retain Subject OE Trt_group VisitDt Visit_Name DIOP_OD DIOP_OS;
set ex (rename=(visitDT = dt));
visitDt = input(dt, missDates.);
run;

proc print data=ex_2;
format visitDt missDates. Trt_group DIOP_OD DIOP_OS missing.;
run;
```

**Output**

| Obs | Subject | OE | Trt_group | VisitDt | Visit_Name | DIOP_OD | DIOP_OS |
|---|---|---|---|---|---|---|---|
| 1 | 111 | OD | Not Randomized | 2011-05-19 | Screening | 19.6 | 21.5 |
| 2 | 112 | OD | Not Randomized | 2011-05-25 | Screening | 26.7 | 24.9 |
| 3 | 113 | OS | Not Randomized | 2011-06-03 | Screening | 17.2 | 18.8 |
| 4 | 114 | OS | Not Randomized | 2011-07-03 | Screening | 15.4 | 14.7 |
| 5 | 111 | OD | Not Randomized | 2011-06-29 | Baseline | 21.3 | Not Available |
| 6 | 112 | OD | Not Randomized | 2011-06-30 | Baseline | 34.6 | Not Available |
| 7 | 113 | OS | Not Randomized | 2011-07-07 | Baseline | Not Available | Not Washed Out |
| 8 | 114 | OS | Exit | Exit | Baseline | Not Available | Exit |
| 9 | 111 | OD | 1 | 2011-07-01 | Postop_1 Day | 18.2 | Not Available |

| Obs | Subject | OE | Trt_group | VisitDt | Visit_Name | DIOP_OD | DIOP_OS |
|---|---|---|---|---|---|---|---|
| 10 | 112 | OD | 2 | 2011-07-02 | Postop_1 Day | 24.3 | Not Available |
| 11 | 113 | OS | 2 | 2011-07-09 | Postop_1 Day | Not Available | 20.8 |
| 12 | 114 | OS | Exit | Exit | Postop_1 Day | Not Available | Exit |
| 13 | 111 | OD | 1 | Lost to follow up | Postop_1 Month | Lost To Follow Up | Not Available |
| 14 | 112 | OD | 2 | 2011-07-28 | Postop_1 Month | 21.8 | Not Available |
| 15 | 113 | OS | 2 | 2011-08-06 | Postop_1 Month | Not Available | 18.1 |
| 16 | 114 | OS | Exit | Exit | Postop_1 Month | Not Available | Exit |
| 17 | 111 | OD | 1 | Lost to follow up | Postop_12 Month | Lost To Follow Up | Not Available |
| 18 | 112 | OD | 2 | Not Done | Postop_12 Month | . | Not Available |
| 19 | 113 | OS | 2 | 2012-06-20 | Postop_12 Month | Not Available | 17.9 |
| 20 | 114 | OS | Exit | Exit | Postop_12 Month | Not Available | Exit |

Taking advantage of special characters in representing missing values not only benefits us on clearly viewing the missing reasons but also provides a convenient way of retrieving a certain subset of missing values from the whole dataset.

```
data sub_LTFU;
set ex_2;
if DIOP_OD = .L or DIOP_OS = .L;
run;

proc print data=sub_LTFU;
format visitDt missDates. Trt_group DIOP_OD DIOP_OS missing.;
run;
```

**Output**

| Obs | Subject | OE | Trt_group | VisitDt | Visit_Name | DIOP_OD | DIOP_OS |
|---|---|---|---|---|---|---|---|
| 1 | 111 | OD | 1 | Lost to follow up | Postop_1 Month | Lost To Follow Up | Not Available |
| 2 | 111 | OD | 1 | Lost to follow up | Postop_12 Month | Lost To Follow Up | Not Available |

## MULTILABEL OPTION ALLOWS DUPLICATES AND OVERLAPPING RANGES IN FORMAT

The MULTILABEL option was added into PROC FORMAT since SAS® 8. It allows each data point to have more than one labels which facilitates frequency calculation and data summary. Here's an example. Suppose we have a dataset called MEDS looks like the following, now we want to know how many subjects took Beta-Blocker, Prostaglandin Analog, Hyperosmotic Agent and Carbonic Anhydrase Inhibitor. MedB is a combination of Beta-Blocker and Prostaglandin Analog. Without using MULTILABEL we will need to generate an indicator for each drug class and calculate the meds frequency based on the drug classes. MULTILABEL options allows less coding.

| Subject | Meds | Drug_Class |
|---|---|---|
| 1 | MedA | Beta-Blocker |
| 1 | MedB | Beta-Blocker and Prostaglandin Analog |
| 2 | MedC | Prostaglandin Analog |
| 2 | MedD | Hyperosmotic Agent |
| 2 | MedA | Beta-Blocker |
| 2 | MedF | Carbonic Anhydrase Inhibitor |
| 3 | MedA | Beta-Blocker |

Using MULTILABEL option:

```
PROC FORMAT;
   VALUE $MEDFMT (MULTILABEL)
   'MedA' =   'Beta-Blocker'
   'MedB' =   'Beta-Blocker'
   'MedB' =    'Prostaglandin Analog'
   'MedC' =   'Prostaglandin Analog'
   'MedD' =   'Hyperosmotic Agent'
   'MedF' =    'Carbonic Anhydrase Inhibitor'
   ;
RUN;

PROC MEANS DATA=MEDS NOPRINT;
CLASS MEDS/ MLF ;
OUTPUT OUT=FREQ_OUT (DROP=_TYPE_ _FREQ_)
N = COUNT;
FORMAT MEDS $MEDFMT.;
RUN;

DATA FREQ_OUT_2;
SET FREQ_OUT;
IF _N_=1 THEN DELETE;
RUN;

PROC PRINT DATA=FREQ_OUT_2 NOOBS;
RUN;
```

The output looks like:

| Meds | COUNT |
|---|---|
| Beta-Blocker | 4 |
| Prostaglandin Analog | 2 |
| Hyperosmotic Agent | 1 |
| Carbonic Anhydrase Inhibitor | 1 |

Note that to make MULTILABEL work we will need to specify MLF option in CLASS statement under PROC MEANS to inform SAS the format is MULTILABEL format. Without specifying this MLF option the output will be looking like the following in which only the first label (‘Beta-Blocker’) will be taken as the label for MedB.

| Meds | COUNT |
|---|---|
| Beta-Blocker | 4 |
| Prostaglandin Analog | 1 |
| Hyperosmotic Agent | 1 |
| Carbonic Anhydrase Inhibitor | 1 |

In the above example, MULTILABEL option is used to allow one single data point being mapped to multiple label values. Another usage of MULTILABEL option is to allow overlapping ranges in a format. Using MEDS_DOSE dataset as an example, suppose we want to know how many medications have a dose under 50mg, under 75mg, under 100mg and above 100mg.

```
PROC FORMAT;
  VALUE DOSE_M (MULTILABEL)
       LOW - 49.99 = 'UNDER 50MG'
       LOW - 74.99 = 'UNDER 75MG'
       LOW - 99.99 = 'UNDER 100MG'
       100.00 - HIGH = '100MG AND ABOVE'
       ;
RUN;


PROC MEANS DATA=MEDS_DOSE NOPRINT;
CLASS DOSE / MLF;
VAR DOSE;
OUTPUT OUT=DOSE_LEVEL_SUMMARY (DROP=_TYPE_ _FREQ_)
       N=N;
FORMAT DOSE DOSE_M.;
RUN;
```

```
DATA DOSE_SUMMARY_2;
SET DOSE_SUMMARY;
IF _N_=1 THEN DELETE;
RUN;

PROC PRINT DATA=DOSE_SUMMARY_2 NOOBS;
RUN;
```

The output looks like so:

| DOSE | **N** |
|---|---|
| 100 AND ABOVE | 1 |
| UNDER 100MG | 3 |
| UNDER 50MG | 2 |
| UNDER 75MG | 2 |

The counts (N) are correct but the order of format labels is obviously not what we want. It appears that SAS sorts the format labels by value. In order to retain the order of the label values that we specified in PROC FORMAT, NOTSORTED option needs to be added. Note that NOTSORTED option does not really mean ‘unsorted’. Instead, it forces the order of the data in the output to be the same as is specified in the PROC FORMAT rather than the default order that SAS provides (either ascending or descending). In this example, in order to get NOTSORTED option work, another option PRELOADFMT and ORDER=DATA must be added in the CLASS statement at the same time. The PRELOADFMT tells SAS to preload all the formats for the class variable before the procedure is executed. Without PRELOADFMT and ORDER=DATA, SAS will ignore the NOTSORTED option and the order of the data in the output will be the same as the default.

```
PROC FORMAT;
  VALUE DOSE_M_N (MULTILABEL NOTSORTED)
  LOW - 49.99 = 'UNDER 50MG'
  LOW - 74.99 = 'UNDER 75MG'
  LOW - 99.99 = 'UNDER 100MG'
  100.00 - HIGH = '100 AND ABOVE'
  ;
RUN;

PROC MEANS DATA=MEDS_DOSE NOPRINT;
CLASS DOSE / MLF PRELOADFMT ORDER=DATA;
VAR DOSE;
OUTPUT OUT=DOSE_LEVEL_SUMMARY (DROP=_TYPE_ _FREQ_)
       N=N;
FORMAT DOSE DOSE_M.;
RUN;
```

## TWO WAYS TO MODIFIY YOUR INFORMATS AND FORMATS

Formats and informats may need to be modified. For example, as a clinical trial goes on information of more and more follow-up visits will be available in the database. Suppose we want to update the $VISIT_NO informat by adding a row for '12 MONTH', here's what we can do:

```
PROC FORMAT;
  INVALUE $VISIT_NO_B
     6 = '12 MONTH'
  OTHER = [VISIT_NO.]
  ;
RUN;
```

As you can see it is very simple. Just add whatever you like in the new informat or format and let OTHER = [the old informat or format]. You may change any item in the old informat in a similar way. For example, change the visit number for Screening from -2 to -3:

```
PROC FORMAT;
  INVALUE $VISIT_NO_C
     -3 = 'SCREENING'
  OTHER = [VISIT_NO.]
  ;
RUN;
```

Another way to modify your informats or formats is to convert them into a dataset, edit the dataset according to your needs and then convert the dataset back to informats or formats. We look at this next.

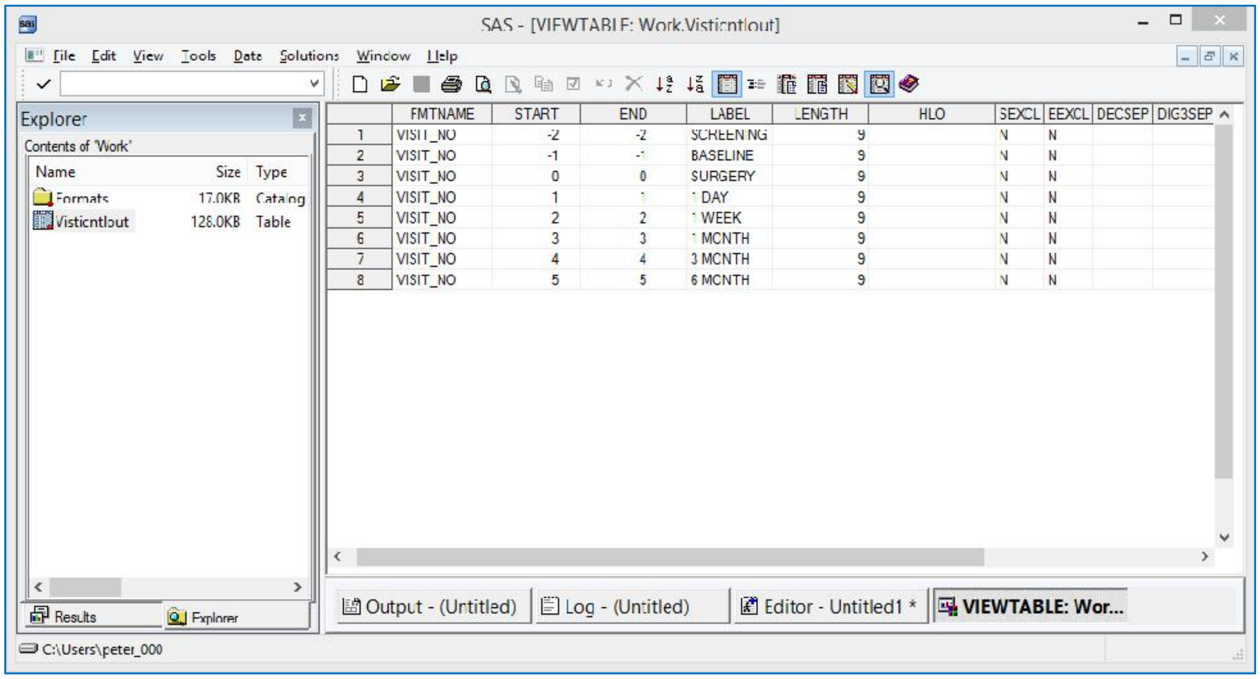
## CREATING FORMATS FROM DATA

It is common that we already have the data for our formats in a table; for example we may already have a table with MEDS with the name, drug class, standard dosage etc.. We can always use SQL or a DATA step merge to add the description to the data, but this would result in a new variable(s) as well as extra processing. A better approach is to create the formats from the data. To do this we use the CNTLIN= option of PROC FORMAT to specify the input data.

Although we can use a dataset as input to PROC FORMAT, not just any dataset can be used; the data needs to follow a specific, but simple, structure. A CNTLIN dataset can have many variables, to best see a complete set of variables use the sister option CNTLOUT= of PROC FORMAT to see the layout of your existing formats.

```
PROC FORMAT CNLTOUT=visitCntlout;
run;
```

If our format catalogue only had the ***VISIT_NO*** format shown earlier, the output would look like (note some columns removed for clarity):



| Name | Size | Type |
|---|---|---|
| Formats | 17.0KB | Catalog |
| Visitcntlout | 128.0KB | Table |

| | FMTNAME | START | END | LABEL | LENGTH | HLO | SEXCL | EEXCL | DECSEP | DIG3SEP |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | VISIT_NO | -2 | -2 | SCREENING | 9 | | N | N | | |
| 2 | VISIT_NO | -1 | -1 | BASELINE | 9 | | N | N | | |
| 3 | VISIT_NO | 0 | 0 | SURGERY | 9 | | N | N | | |
| 4 | VISIT_NO | 1 | 1 | 1 DAY | 9 | | N | N | | |
| 5 | VISIT_NO | 2 | 2 | 1 WEEK | 9 | | N | N | | |
| 6 | VISIT_NO | 3 | 3 | 1 MONTH | 9 | | N | N | | |
| 7 | VISIT_NO | 4 | 4 | 3 MONTH | 9 | | N | N | | |
| 8 | VISIT_NO | 5 | 5 | 6 MONTH | 9 | | N | N | | |

A CNTLOUT dataset (only selected columns)

To create the ***VISIT_NO*** format from data the minimum we need to provide is:

- FMTNAME – the name of the format
- START - the starting value
- LABEL - the label

The code to create this could look like:

```
data visitCntlIn;
1      retain fmtname 'VISIT_NO';
       infile DATALINES end=done truncover;
2        input start 2. +1 label $9.;
         output;
DATALINES;
-2 SCREENING
-1 BASELINE
 0 SURGERY
 1 1 DAY
 2 1 WEEK
 3 1 MONTH
 4 3 MONTH
 5 6 MONTH
 ;;
 run;
PROC FORMAT CNTLIN=visitCntlIN;
run;
```

1. We RETAIN the format name so it will be available to all rows

2. We read in the variables START and LABEL

In this simple example we read the data instream so we were able to create the variables names we needed; if you already have a dataset with **VISIT_NO** and **VISIT** you would need to do some simple processing:

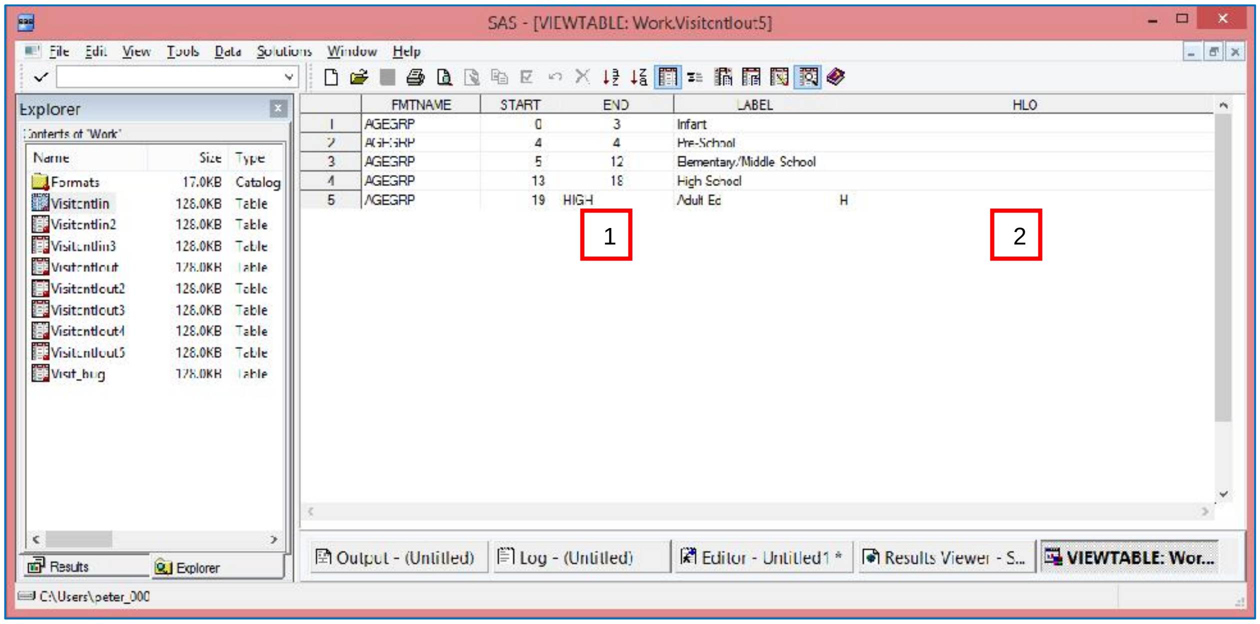
```
DATA visitCntlin;
1     retain fmtName ‘VISIT_NO’;
2     set visitDescriptions (rename=(visit_no = start visit=label));
      output;
Run;
```

1. Create the format name and keep it for all rows.
2. When reading in the data, use the dataset RENAME= option;

If your formatted values covers a range as follows:

```
PROC FORMAT;
     VALUE AGEGRP
 0 - 3    = 'Infant'
 4        = 'Pre-School'
 5 - 12   = 'Elementary/Middle School'
13 - 18   = 'High School'
19 - high = 'Adult Ed'
;
run;
```

then you need to add two variables , ***END*** and ***HLO***,  to your CNTLIN data. The dataset would look like:



| Name | Size | Type |
|---|---|---|
| Formats | 17.0KB | Catalog |
| Visitcntlin | 128.0KB | Table |
| Visitcntlin2 | 128.0KB | Table |
| Visitcntlin3 | 128.0KB | Table |
| Visitcntlout | 128.0KB | Table |
| Visitcntlout2 | 128.0KB | Table |
| Visitcntlout3 | 128.0KB | Table |
| Visitcntlout4 | 128.0KB | Table |
| Visitcntlout5 | 128.0KB | Table |
| Visit_bug | 128.0KB | Table |

| | FMTNAME | START | END | LABEL | HLO |
|---|---|---|---|---|---|
| 1 | AGEGRP | 0 | 3 | Infant | |
| 2 | AGEGRP | 4 | 4 | Pre-School | |
| 3 | AGEGRP | 5 | 12 | Elementary/Middle School | |
| 4 | AGEGRP | 13 | 18 | High School | |
| 5 | AGEGRP | 19 | HIGH | Adult Ed | H |

1. END – note that for age 4, end and start are the same.
2. HLO – for the last entry (19 to HIGH) the HLO variable is ‘***H***’

As your formats become more complex you can start to take advantage of the other columns in the CNTLIN dataset. A best practice is to manually create PROC FORMAT code that has the features you want to include in your data driven format (eg, using missing, LOW, OTHER, less than/less than equal ranges), generate the CNTLOUT dataset to see how the variables have been set, then create code to mimic this.

## FUNCTIONS AND FORMATS

Starting in SAS 9.3, we can incorporate user written functions into PROC FORMAT. Before we do that we will do a short review of writing your own formats with PROC FCMP. For a more complete introduction to PROC FCMP see Eberhardt 2009, 2010.

### PROC FCMP

PROC FCMP allows us to create functions and call routines in DATA step language; these functions and call routines can be used anywhere the built-in SAS functions and call routines can be used. Creating your own functions allows you to create more modular and robust SAS programs.

A common problem when dealing with global data is the need to convert units of measure from the global standard metric to the more archaic imperia; for example from centimeters to inches, to degrees C to degrees F. A simple set of functions to perform these coversions are:

```
1 proc fcmp outlib=work.conv.base ;

2 FUNCTION C_to_F(C) ;
    /* convert degrees fahrenheit to degrees celsius */
3   return (32 + (C *  (9. / 5.)));

4 ENDSUB;

  FUNCTION IN_to_CM(inch) ;
    return (inch * 2.54);
  ENDSUB;

  FUNCTION F_to_C(F) ;
    /* convert degrees fahrenheit to degrees celsius */
    return ((F - 32) * (5. / 9.));
  ENDSUB;

  FUNCTION CM_to_IN(cm) ;
    return (cm * 0.3937008) ;
  ENDSUB;

  run;
  quit;
```

| | | |
|---|---|---|
| 1. | PROC FCMP | – write the functions to a WORK library |
| 2. | FUNCTION C_to_F ( C ) | – create a function with one input argument |
| 3. | RETURN | – the value to be returned to the calling program follows the return statement; |
| 4. | ENDSUB | - ends the function definition |

As we see, many functions can be defined and made available. The following shows how to use the functions:

```
/* tell SAS where to find your functions */
options cmplib=work.conv;
```

```
data testDS_1;
    format inch cm 6.2;
    do inch = 0 to 100 by 0.5;
2         cm = in_to_cm(inch);
         output;
    end;
run;
```

1. options cmplib=work.conv – tells SAS where to find your functions
2. cm = in_to_cm(inch); - invoke the function. Looks the same as a SAS function

Although creating your own functions is valuable, the question is "How do they relate to formats??". The answer is simple: they become part of the format definition. First, look at the PROC FORMAT code:

```
proc format;
1 alue INtoCM  (default=5)
2 ther = [in_to_cm()]

value CMtoIN  (default=5)
other = [cm_to_in()]
;
run;
```

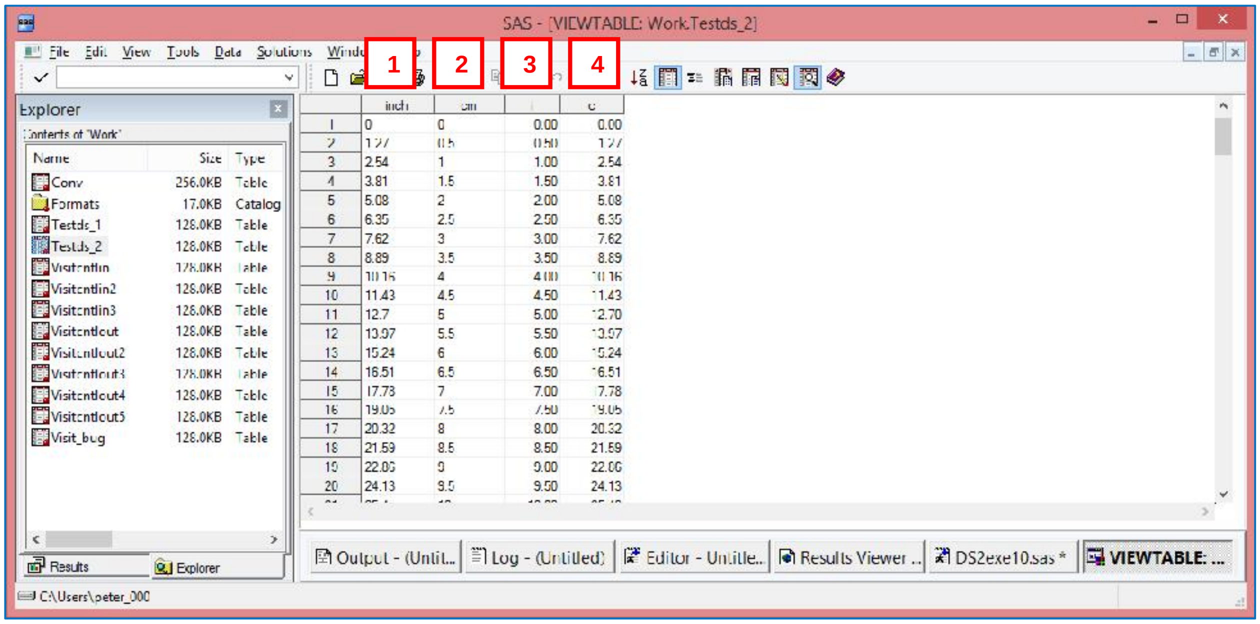
1. default = 5 - sets a default width
2. [in_to_cm()] - tell FORMAT to use the in_to_cm function to determine the contents of the label
   - NOTE: the function specified can take only one argument
   - Since the compiler does not know what may be returned from the function to set a width for the format, we use the default= to give a width

The following code shows how these formats may be used:

```
data testDS_2;
1      format inch inTOcm.  /* store as inch, display as cm */
             cm   cmTOin.  /* store as cm, display as inch */
2            i c 6.2       /* equavalent in own units      */
          ;
      do inch = 0 to 100 by 0.5;
3        i = inch;
         cm = in_to_cm(inch); /* convert to cm          *
4        c = cm;
         output;
      end;
run;
```

1. Format the inch variable as inTOcm, will display as cm
2. Format the cm variable as cmTOin, will display as inch
3. Keep an unformatted value if inch (i) and cm ( c )
4. Create a new variable, converting the inch value to cm

The results are:



SAS - [VIEWTABLE: Work.Testds_2]

Explorer — Contents of 'Work'

| Name | Size | Type |
|---|---|---|
| Conv | 256.0KB | Table |
| Formats | 17.0KB | Catalog |
| Testds_1 | 128.0KB | Table |
| Testds_2 | 128.0KB | Table |
| Visitcntlin | 128.0KB | Table |
| Visitcntlin2 | 128.0KB | Table |
| Visitcntlin3 | 128.0KB | Table |
| Visitcntlout | 128.0KB | Table |
| Visitcntlout2 | 128.0KB | Table |
| Visitcntlout3 | 128.0KB | Table |
| Visitcntlout4 | 128.0KB | Table |
| Visitcntlout5 | 128.0KB | Table |
| Visit_bug | 128.0KB | Table |

| | inch | cm | i | c |
|---|---|---|---|---|
| 1 | 0 | 0 | 0.00 | 0.00 |
| 2 | 1.27 | 0.5 | 0.50 | 1.27 |
| 3 | 2.54 | 1 | 1.00 | 2.54 |
| 4 | 3.81 | 1.5 | 1.50 | 3.81 |
| 5 | 5.08 | 2 | 2.00 | 5.08 |
| 6 | 6.35 | 2.5 | 2.50 | 6.35 |
| 7 | 7.62 | 3 | 3.00 | 7.62 |
| 8 | 8.89 | 3.5 | 3.50 | 8.89 |
| 9 | 10.16 | 4 | 4.00 | 10.16 |
| 10 | 11.43 | 4.5 | 4.50 | 11.43 |
| 11 | 12.7 | 5 | 5.00 | 12.70 |
| 12 | 13.97 | 5.5 | 5.50 | 13.97 |
| 13 | 15.24 | 6 | 6.00 | 15.24 |
| 14 | 16.51 | 6.5 | 6.50 | 16.51 |
| 15 | 17.78 | 7 | 7.00 | 17.78 |
| 16 | 19.05 | 7.5 | 7.50 | 19.05 |
| 17 | 20.32 | 8 | 8.00 | 20.32 |
| 18 | 21.59 | 8.5 | 8.50 | 21.59 |
| 19 | 22.86 | 9 | 9.00 | 22.86 |
| 20 | 24.13 | 9.5 | 9.50 | 24.13 |

1. The inch value displayed as cm
2. The cm value displayed as inch
3. The underlying value in inches
4. The underlying value in cm

Now that we have started to master the use of functions in formats, let's extend this by introducing globalization using LOCALE. For more on globalization and LOCALE see Qin 2014

## LOCALE

LOCALE is a feature of SAS that allows us to have language specific features available to us without having to have different code bases; we only need to specify a LOCALE option. PROC FORMAT was updated to be LOCALE aware, so we can create formats/informats that are language specific. No matter the language that we wish to use, the format name is the same. Simple examples usually show something like:

```
1   options locale=en_us;

2   PROC FORMAT locale library=work.fmts;
        value yn
3       1 = 'Yes'
        2 = 'No'
  ;
  run;

4   options locale=FR_FR;

    PROC FORMAT locale library=work.fmts;
        value yn
        1 = 'Oui'
        2 = 'Non'
  ;
  run;
```

```
5  options fmtsearch=(work.fmts/locale);
6 options locale=en_us;

DATA _null_;
     do i = 1 to 2;
7        put i= i= yn.;
       end;
run;

8  options locale=FR_FR;

DATA _null_;
     do i = 1 to 2;
9        put i= i= yn.;
       end;
run;
```

1. Set the LOCALE to US English
2. When using LOCALE specific formats a library must also be specified. SAS will create format catalogue called WORK.fmts_en_us
3. Create formats that are specific to US English
4. Repeat the three steps above for the second LOCALE, French
5. Specify the search path for format libraries
6. Reset the LOCALE to US English
7. Display values to the log (see below)
8. Reset the LOCALE to French
9. Display values to the log (see below)

Note that the DATA step did not change, although the results printed are different:

US English:

```
1419
1420  options fmtsearch=(work.fmts/locale);
1421  options locale=en_us;
1422  DATA _null_;
1423       do i = 1 to 2;
1424          put i= i= yn.;
1425      end;
1426  run;

i=1 i=Yes
i=2 i=No
```

French:

```
1427
1428  options locale=FR_FR;
1429  DATA _null_;
1430       do i = 1 to 2;
1431          put i= i= yn.;
1432      end;
```

```
1433  run;

i=1 i=Oui
i=2 i=Non
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

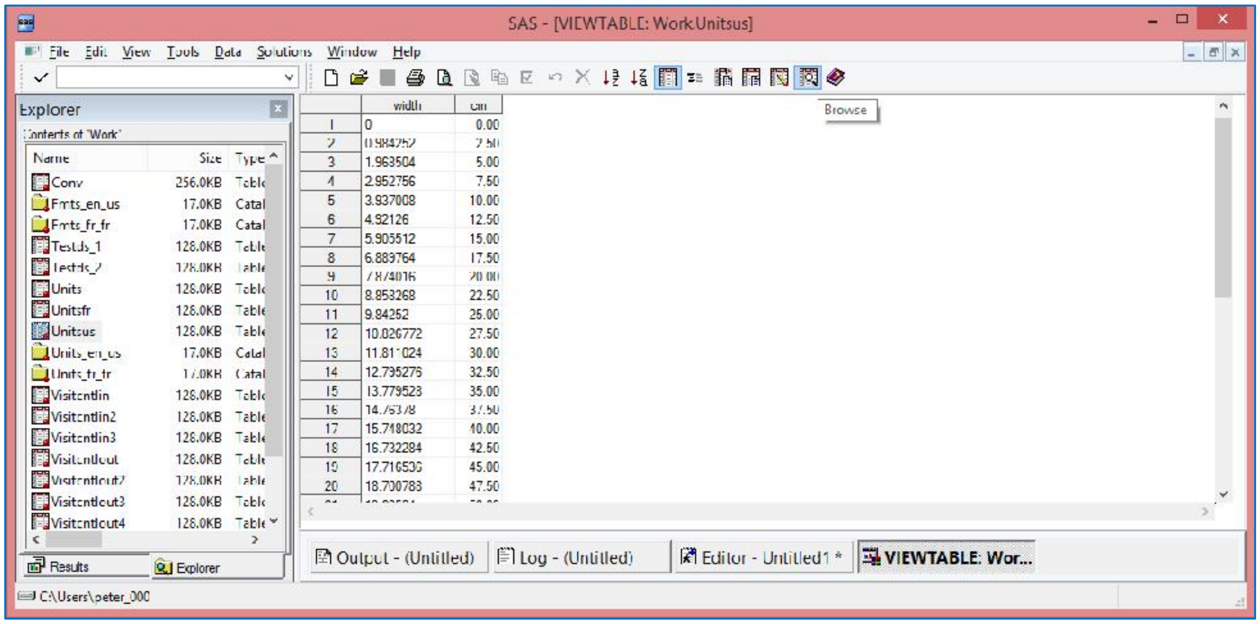
Having seen how we can use functions in our formats, in the examples above to display units in a country specific manner, we can quickly see how to combine LOCALE and functions to easily show our units in a LOCALE specific manner. Here we will augment out ***cmTOin*** format to be locale specific. In this example, we are assuming the data are stored in the international standard of metric measurments:

```
1   options locale=en_us;

proc format locale library=work.units
      value useLen
2     other = [cm_to_in()]
       ;
run;

3   options locale=fr_fr;
proc format locale library=work.units;
      value useLen
4     other = [6.2]
       ;
run;

5    options fmtsearch=(work.units/locale);

6   options locale=en_us;

data unitsUS;
7    format width useLen. cm 6.2;
     do width = 0 to 100 by 2.5;
         cm = width;
         output;
       end;
run;

6 options locale=fr_fr;
data unitsFR;
     format width useLen. cm 6.2;
     do width = 0 to 100 by 2.5;
         cm = width;
         output;
       end;
run;
```

1. Set the LOCALE to US English

2. Create the format using the cm_to_in() function. Remember, the data are stored as the metric value but we want to display in the imperial measure
3. Set the LOCALE to French
4. Do not convert the value, simply apply the numeric format 6.2
5. Set the format search path. This is necessary so SAS can find out locale specific formats.
6. Set the LOCALE to US English
7. Use the format useLen. which will display the metric measures as inches
8. Repeat for French.

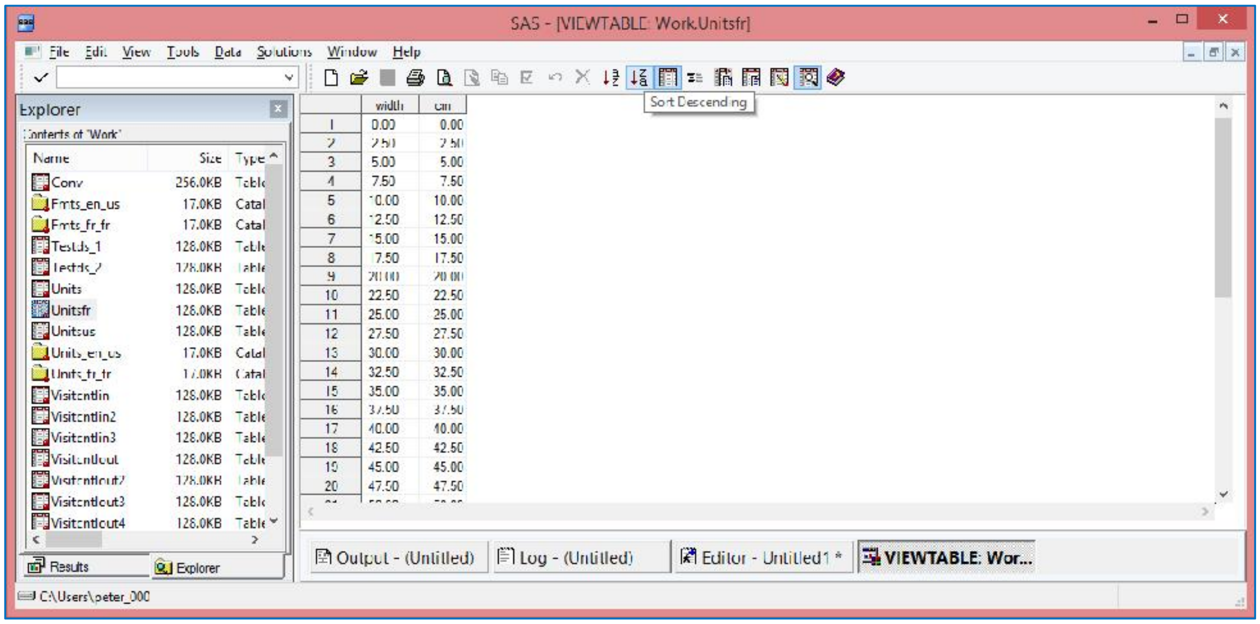
Partial output from unitsUS:



| Name | Size | Type |
|---|---|---|
| Conv | 256.0KB | Table |
| Fmts_en_us | 17.0KB | Catal |
| Fmts_fr_fr | 17.0KB | Catal |
| Testds_1 | 128.0KB | Table |
| Testds_2 | 128.0KB | Table |
| Units | 128.0KB | Table |
| Unitsfr | 128.0KB | Table |
| Unitsus | 128.0KB | Table |
| Units_en_us | 17.0KB | Catal |
| Units_fr_fr | 17.0KB | Catal |
| Visitcntlin | 128.0KB | Table |
| Visitcntlin2 | 128.0KB | Table |
| Visitcntlin3 | 128.0KB | Table |
| Visitcntlout | 128.0KB | Table |
| Visitcntlout2 | 128.0KB | Table |
| Visitcntlout3 | 128.0KB | Table |
| Visitcntlout4 | 128.0KB | Table |

| | width | cm |
|---|---|---|
| 1 | 0 | 0.00 |
| 2 | 0.984252 | 2.50 |
| 3 | 1.968504 | 5.00 |
| 4 | 2.952756 | 7.50 |
| 5 | 3.937008 | 10.00 |
| 6 | 4.92126 | 12.50 |
| 7 | 5.905512 | 15.00 |
| 8 | 6.889764 | 17.50 |
| 9 | 7.874016 | 20.00 |
| 10 | 8.858268 | 22.50 |
| 11 | 9.84252 | 25.00 |
| 12 | 10.826772 | 27.50 |
| 13 | 11.811024 | 30.00 |
| 14 | 12.795276 | 32.50 |
| 15 | 13.779528 | 35.00 |
| 16 | 14.76378 | 37.50 |
| 17 | 15.748032 | 40.00 |
| 18 | 16.732284 | 42.50 |
| 19 | 17.716535 | 45.00 |
| 20 | 18.700788 | 47.50 |

Partial output from unitsFR:



| Name | Size | Type |
|---|---|---|
| Conv | 256.0KB | Table |
| Fmts_en_us | 17.0KB | Catal |
| Fmts_fr_fr | 17.0KB | Catal |
| Testds_1 | 128.0KB | Table |
| Testds_2 | 128.0KB | Table |
| Units | 128.0KB | Table |
| Unitsfr | 128.0KB | Table |
| Unitsus | 128.0KB | Table |
| Units_en_us | 17.0KB | Catal |
| Units_fr_fr | 17.0KB | Catal |
| Visitcntlin | 128.0KB | Table |
| Visitcntlin2 | 128.0KB | Table |
| Visitcntlin3 | 128.0KB | Table |
| Visitcntlout | 128.0KB | Table |
| Visitcntlout2 | 128.0KB | Table |
| Visitcntlout3 | 128.0KB | Table |
| Visitcntlout4 | 128.0KB | Table |

| | width | cm |
|---|---|---|
| 1 | 0.00 | 0.00 |
| 2 | 2.50 | 2.50 |
| 3 | 5.00 | 5.00 |
| 4 | 7.50 | 7.50 |
| 5 | 10.00 | 10.00 |
| 6 | 12.50 | 12.50 |
| 7 | 15.00 | 15.00 |
| 8 | 17.50 | 17.50 |
| 9 | 20.00 | 20.00 |
| 10 | 22.50 | 22.50 |
| 11 | 25.00 | 25.00 |
| 12 | 27.50 | 27.50 |
| 13 | 30.00 | 30.00 |
| 14 | 32.50 | 32.50 |
| 15 | 35.00 | 35.00 |
| 16 | 37.50 | 37.50 |
| 17 | 40.00 | 40.00 |
| 18 | 42.50 | 42.50 |
| 19 | 45.00 | 45.00 |
| 20 | 47.50 | 47.50 |

Note the width in the US English table is displayed in inches, although the underlying data are cm. In the French table, the displayed value has not changed.

Also note that, in general, the LOCALE value will be set in a configuration file, or possibly an autoexec, to once the format catalogues are created and stored in permanent libraries, no change has to be made to the code to get the appropriate display values.

From this simple example you can also see how INFORMATS can be generated to convert raw data that would come from different countries that use different measurement scales into a common unit. This will be left for you to practice in the safety of your own office!!

## CONCLUSION

We looked at several examples of PROC FORMAT, each of which should get you started on the journey to be creative in your use of formats and informats

## REFERENCES

Carpenter, Arthur, “Building and Using User Defined Formats” *Proceedings of the 29th Annual SAS User Group International Conference*

Chakravarthy, Venky “MULTILABEL - A useful addition to the FORMAT procedure” *Proceedings of the 2004 PharmaSUG Conference*.

Christen, Erin “ PROC FORMAT is Our Friend”, *Proceedings of the 2004 PharmaSUG Conference.*

Eberhardt, Peter “A Cup of Coffee and PROC FCMP: I Cannot Function Without Them”, *Proceedings of the SAS Global Forum 2009 Conference*. Cary, NC: SAS Institute Inc.

Eberhardt, Peter “Functioning at an Advanced Level: PROC FCMP and PROC PROTO”, *Proceedings of the SAS Global Forum 2010 Conference*. Cary, NC: SAS Institute Inc.

Qin, XiaoJin “The SAS® LOCALE Option:Win an international passport for your SAS code”, Proceedings of the 2014 PharmaSUG Conference.

## CONTACT INFORMATION

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

| | |
|---|---|
| Name: | Peter Eberhardt |
| Enterprise: | Fernwood Consulting Group Inc. |
| City, Prov: | Toronto, ON, Canada |
| E-mail: | peter@fermwood.ca |
| Twitter: | rkinRobin |

| | |
|---|---|
| Name: | Lucheng Shao |
| Enterprise: | Ivantis Inc |
| City, Prov: | Irvine, CA, U.S |
| E-mail: | LShao@ivantisinc.com |