Variable Labels

Good programming style veritably requires the labeling of variables in SAS® data sets. Not only does an adequate label tell more about the variable than its eight-character name does, but that label can be used by numerous SAS procedures to identify what the variable represents in the analysis. PROC PRINT, for example, can easily be used to prototype -- or complete -- most forms of reports, using perhaps temporary labels to neatly identify the columns (refer to the PROC PRINT LABEL option for details). There may even be room in the forty-character label to explain what the variable values mean (so a SAS format isn't needed to do that) or when the variable was last updated, or briefly how or why it was created. Labels help document the information, so others with access to the data -- your users, your successors, or other partners in crime -- may immediately have a better understanding of the variables in the data set or maybe even how the data set relates to other data sets in a system. Even the person who created the variable can profit if he or she needs to return to a file created in the murky past only to forget the quirks or meaning of some or all of the variables? Variable labels can help to jog those memories, yielding a more timely and productive familiarity with the data. It's hard to argue with the need to label variables unless time or other constraints absolutely will not permit it.

Merging SAS data sets where like variables are present can have some interesting results. Users with at least some experience with MERGE using a BY statement know that, when merging two or more data sets, each with one or more variables in common -- other than the BY variables themselves, of course -- an interesting thing happens. For those common variables, SAS keeps the value from the data set named last in the MERGE statement, even if that value is missing. One wonders where the data went, until realizing that the culprit is a variable in the last-named data set, the value of which is sometimes -- or maybe always -- missing.

Unlike merging data however, SAS follows a different rule for variable labels during a merge. SAS' "program data vector" or PDV, which contains information about the variables in a DATA step, such as variable name, type, length, format, and whether the variable is to be kept or dropped, also keeps a label if one is available. But it is satisfied with the first non-blank label it gets for a variable, regardless of the label in any subsequently-named data set. (A LABEL statement, of course, can override any available label.) So there is a hierarchical order demonstrated here:

1) The LABEL statement takes precedence.
2) The program data vector prefers non-blank labels to blank ones.
3) The PDV keeps the first label it receives.

How can you use this to your advantage? Let's say you have a data set that has numerous neatly-labeled variables, and one with many of the same variables, but which is not labeled. You can set or "merge" zero observations from the neatly-labeled data set, perhaps keeping only specific variables, with the data set you wish to label. No LABEL statement is required, no observations from the labeled data set contaminate the output data set, and a minimum of typing is required. It can be done swiftly and interactively. A complete example is shown in Figure 1. Note that data set OLD.LABELS has the
preferred variable labels; data set
UNLABELLD has poor or no variable labels,
and the resulting data set, LABELED, will
also have the preferred labels.
Conceivably, a KEEP list should be
provided when SETting OLD.LABELS,
depending on the application.

DATA LABELED;
  SET OLD.LABELS (OBS=0)
RUN;

Figure 1: Applying Existing Variable
Labels to a New Data Set.

Here's an extension of the technique:
SAS under DOS often has only limited
memory available to it -- perhaps not
enough memory to load the PDV plus data
values plus software. One solution: keep
one data set with complete data and all
observations and keep the labels in
another data set containing no
observations. PROC CONTENTS can be
used on the labeled data set to provide
documentation (but without the number of
observations in the complete data set).
When variable labels are needed or
preferred, simply SET or MERGE both data
sets, wisely keeping only the variables
required for the analysis or report. It works,
and can make a 640K machine suffice
where expensive expanded memory is not
available.

Another example of using variable labels,
this time to improve report legibility, is
shown in Figure 2. The top portion of the
figure illustrates the default use of variable
labels when employing PROC PRINT with
a BY statement. Note that the information
is tightly spaced, sometimes to the point of
near illegibility. By specifying labels which
contain leading blanks for the BY
variables, the BY variables are segregated
for printing, making the BY-groups much
easier to identify without reading the entire
line SAS prints at the beginning of a BY
group. The more BY variables used, the
more important this technique becomes to
avoid a confusing display of information.

PROC PRINT, STANDARD LABELED BY-VARIABLES
  COUNTY=MODOC SALESPERSON=TAYLOR
  OBS KSALESMN
  1  90
PROC PRINT, SPECIALLY-LABELED BY-VARIABLES
  COUNTY=MODOC SALESPERSON=CROSBY
  OBS KSALESMN
  2  80

Figure 2: Using Variable Labels to
Improve PROC PRINT BY-group
Legibility

Other applications for manipulation of
labels exist, to save typing and ensure
consistency, but are needed less often,
and are beyond the scope of this paper.

Data Set Labels

Ever wonder how some people arrive at
the names they use for their SAS data
sets? Generally, there's a reason,
however indefensible, but it's often not
intuitively obvious where the name came
from, or what the data set contains, even
with reports from PROC CONTENTS and
PROC PRINT available. Who hasn't
named a data set MERGED or SORTED or
the like? Well, SAS provides a method to
label data sets too. Providing those labels
means following the same rules as coding
variable labels (forty characters max,
embedded quotes can be tricky, macro
variable substitution is permissible, etc.).
Wow! Think of it! A label can describe
how the data set was created, which
version it is, what twists were applied in
creating it, who did it, what subset of a
population it represents, etc. It can even
reveal the sort order of a data set, a
popular request on the annual SAS
Software Ballot. Now researchers don't
have to guess at the sort order (assuming
the label is correct), unnecessary sorts
need not be done (precluding the
inevitable "I'll sort it just in case"), and the
data set is better documented in the
bargain.
There are two ways to assign the label to the data set and two ways to view it. The data set LABEL statement is illustrated in Figure 3, first for a DATA step, then for a PROC SORT step, then for a PROC SUMMARY step. I deliberately chose funky names (data sets should almost never be called "MERGED," but that helps highlight the usefulness of data set labels. The other way to add -- or change -- a label is with our friend PROC DATASETS. I usually use it interactively, but code to assign a label with PROC DATASETS in batch is shown in Figure 4. PROC DATASETS in interactive, full-screen mode also allows one to view a data set label, as does PROC CONTENTS in interactive or batch mode, as shown in the output in Figure 5.

```
DATA MERGED (LABEL=RECENT UPDATES SORTED BY DATE);
   MERGE MARCH.SALES (IN = TARGET)
        MASTER.FILE;
   BY DATE;
   IF TARGET;
RUN;

PROC SORT
   DATA=MERGED
   OUT=SORTED (LABEL=SORTED BY 10 NUMBER);,
   BY 10;
RUN;

PROC SUMMARY DATA=SORTED;
   CLASS STATE MONTH;
   VAR KSALES;
   OUTPUT OUT=SUMMARY (LABEL = MEAN SALES BY STATE AND MONTH)
            MEAN=KSALESMN;
   LABEL KSALESMN='MEAN SALES'.
RUN;
```

Figure 3: Using the LABEL parameter to Assign Data Set Labels.

```
PROC DATASETS LIB=WORK;
   MODIFY SUMMARY (LABEL=MEAN SALES IN $000s BY STATE AND MONTH);
   LABEL KSALESMN = 'MEAN SALES IN $000s';
RUN;
```

Figure 4: Using PROC DATASETS in Batch to Assign or Modify Labels.

Variable Lengths

Any SAS programmer worth his or her salt has used the LENGTH statement to define the lengths of numeric and/or character variables. It's often a necessity. But have you noticed that it can affect the order of variables in a SAS data set? Here's the key: SAS generally puts variables in the order in which it first becomes aware of them. This is why unwanted variables -- subscripts or typos, for example -- often appear last in a PROC CONTENTS positional listing. The data set which is SET or MERGED first has its variables come first in the output data set. Any INPUT statement can define the position of variables. And a LENGTH statement, if placed before a SET, MERGE, UPDATE or INPUT statement, will set the order of the variables it specifies.

How can this be used to your advantage? Easy. Maybe variable order is critical. Quick -- how many variables are in the range VAR1--VAR2? You don't know -- maybe two, maybe two thousand. By ordering the variables, the number of variables implied by VAR1--VAR2 can be known, and subsequent coding of ARRAYS, VAR lists, and KEEP and DROP statements can be greatly eased. Series' of variables which are inappropriately

```
CONTENTS PROCEDURE
CONTENTS OF SAS MEMBER WORK.SUMMARY

CREATED BY TSO USERID...
AT 10:15 MONDAY, MARCH 21...
DSNAME=...
BLENDDR...
DATA SET LABEL=MEAN SALES IN $000s BY STATE MEMTYPE=DATA...
```

Figure 5: Data Set Label Displayed by PROC CONTENTS.

Something worth remembering: when a data set is sorted in place or otherwise recreated, it loses its label. That is, the sorted version of the data set is really a new data set. Another tip: only PROC COPY respects the old label when creating a new data set, copying it faithfully.
interspersed with each other, such as time series, can be properly ordered and segregated with **LENGTH**.

Maybe it would be preferable to place key variables first in the data set, if for no other reason than that **PROC PRINT**, **PROC FSEEDIT**, and other easy-to-use reporting mechanisms will then list the keys first. An example of code to do this is shown in Figure 6. Remember, not all the variables must be listed, just the ones desired to come first. **SAS** will fill in the remainder of the PDV as it becomes aware of them.

```
DATA REORDRD;
  LENGTH SSN 5 DATE 3;
  SET OLDORDER;
RUN;
```

Figure 6: Reordering Variables in a **SAS** data set.

The rule for **SETting** or **MERGEing** like variables with different lengths is simple: **SAS** sets up its PDV with the length it encounters first, potentially rendering useless the data from subsequently-named data sets. For example, **SETting** two data sets, both of which contain the character variable **NAME**, would result in truncation if **NAME** were shorter in the first-named data set. Probably not good. Worse, if the data sets contained, for example, a numeric variable **PHONE**, but the first had a length of four bytes (to store seven-digit numbers) and the second had a length of six bytes (to accommodate ten-digit numbers with area codes), the second set of numbers would lose much significance in the output data set, resulting in ten-digit gibberish. The solution: set the **LENGTH** of **NAME** and **PHONE** to their maximums, before the **SET** statement.

Standish W. Sibley
Sibley Enterprises
(415) 824-9331, (415) 973-0819

**SAS** is a registered trademark of **SAS Institute, Inc., Cary, NC, USA.**