ABSTRACT

PROBLEM:
To output for 60 variables, human-readable explanations from a dictionary of 15,000 codes with 35 character definitions.

ORIGINAL SOLUTION:
Formatting via Table Lookup using matchMERGE required 60 sorts, 60 data statements, and 60 merges. Clumsy and costly.

BETTER SOLUTION:
Selecting pertinent definitions only via data search to create formats.

USEFUL SAS FEATURES:
- DO & ARRAY with (*), _ALL_, DIM function
- KEEP and OUTPUT
- PROC SORT with NODUP
- DQUOTE
- matchMERGE
- RETAIN

INTRODUCTION

One advantage of SAS is the ease of outputting human-readable reports from compactly coded data sets. Part numbers may be stored efficiently in a database while widget names and whatchamacallits are output for mortals to peruse.

PROC FORMAT is the workhorse that makes all this plus several other operations possible and simple.

In earlier versions of SAS and with smaller direct access storage devices, it was not possible to use PROC FORMAT for large dictionaries. The alternative was performing a table lookup (TLU) by match-merging.

Such a TLU is accomplished by having the dictionary entries and codes stored in a SAS data set in alphanumeric order. The database is sorted by the same code and the two are merged with a BY statement. This is a useful technique and comes in handy in some situations. Costs mount rapidly, however, when many sorts are performed.

Although the example used here to illustrate this situation pertains to a medical database, the technique is useful in other applications.

Medical problems - diseases, disorders, symptoms, syndromes - are universally described with a 6 character code. In order to meet governmental regulations and especially to receive Medicare reimbursement, hospitals are required to retain up to 60 diagnoses per patient per hospital admission.

Many thousands such problems or diagnoses have been defined. To be intelligible and reasonably descriptive, the English definition of each such diagnosis needs 30 characters. The online dictionary to which I have access has 15000 such definitions. Elements in a PROC FORMAT statement would look like:

```
PROC FORMAT;
VALUE $DIAGNOS
'E804.8'='FALL FROM TRAIN-PERS NEC'
'E805.2'='HIT BY TRAIN-PEDESTRIAN'
'E810.0'='MV-TRAIN COLL-DRIVER'
'E810.1'='MV-TRAIN COLL-PASNGR'
```

Multiplying 35 characters by 15k comes to half a meg - a healthy chunk of space. One might think that space requirements could be reduced by shortening the definition of E805.2 but it is important to retain all these elements for readability as well as completeness. It must be clear that a pedestrian was hit - not a cyclist or a Martian. It makes a difference, too, that the pedestrian was hit by a train, not a rock, space missile, or spear.
Consider a previously created data set DICTNARY with variables CODE and DIAGDEF already sorted by CODE. To format these definitions via a match-merge, the program would be:

**Figure 2**

```sas
DATA PATIENTS;
SET PATIENTS.ONE
(KEEP=PATNAME PATNO DIAG1-DIAG60);
PATIENTS.TWO ...
PROC SORT DATA=PATIENTS;
BY DIAG1;
DATA PATIENTS
(DROP=CODE RENAME=(DIAGDEF=DIAG1));
MERGE DICTNARY PATIENTS
(IN=INPAT RENAME=(DIAG1=CODE));
IF INPAT;
PROC SORT DATA=PATIENTS;
BY DIAG2;
DATA PATIENTS
(DROP=CODE RENAME=(DIAGDEF=DIAG2));
MERGE DICTNARY PATIENTS
(IN=INPAT RENAME=(DIAG2=CODE));
IF INPAT;

*repeat for DIAG3 .... DIAG60;
```

The above should be accomplished with a simple macro, reducing coding but increasing execution time somewhat.

With a format library the code can be reduced to:

**Figure 3**

```sas
DATA INPAT;
SET PERM.PATIENTS
(KEEP=PATNAME PATNO DIAG1-DIAG60);
FORMAT DIAG1-DIAG60 $DIAGNOSES;
```

**PERMANENT FORMAT LIBRARY**

**Figure 4** shows an IBM OS job to create a permanent SAS format library with a subsequent job step using the created format.

**Figure 4**

```sas
//USERIDI JOB SASLIB,etc....
//STEP1 EXEC SAS
//SASLIB DD DSN=USERIDI.SAS.FORMATS,
//UNIT=SYSDA,
//DISP=(NEW,CATLG),
//SPACE=(CYL,(1,1,1))
PROC FORMAT LIBRARY=SASLIB;
VALUE $SEX 'M'='MALE' 'F'='FEMALE';

//STEP2 EXEC SAS
//SASLIB DD DSN=USERIDI.SAS.FORMATS,
//DISP=SHR
DATA A;
  INPUT NAMES SEX$;
  FORMAT SEX $SEX.;
  CARDS;
  MEG F
  CHRIS M
  BETH F
PROC PRINT NOOBS;  *output follows;
  NAME SEX
  MEG FEMALE
  CHRIS MALE
  BETH FEMALE
```

**STEP1** in Figure 4 creates a permanent SAS FORMAT library with a single member: the $SEX format. Virtually any number of other formats can be added to same library. The DDNAME must be SASLIB; that makes it permanent. Succeeding programs use the format library by incorporating the SASLIB DD statement with DISP=SHR and associating the proper format name via a FORMAT statement as in Step 2. A SAS format library is contained in a dedicated POS, unlike other SAS files which are sequential.

**STEP2** uses the $SEX format created previously.

This was accomplished easily enough with only two exes to define but formatting 15,000 diagnoses is another matter. Not only is the quantity great but fewer than 25% of these problems are seen at this large medical center. There have been no recent cases of the black plague, leprosy or smallpox. AIDS, on the other hand, arrived only in 1982 and is not even in the 15K dictionary. Many small aches and pains may be eliminated without running afoul of current governmental regulations. Thus hangnails and funny bone sensations may be purged.

To reduce the format library to a more useful size, the code in Figure 5 checks patients from recent years to form a data set of diagnoses currently being used.
Figure 5 - MAIN PROGRAM

1  OPTION DQUOTE;  *** for clarity in manipulating single quotes;
2  DATA DXCHECK(KEEP=CODE);    SET PATIENTS.ONE(KEEP=DIAG1-DIAG60)
3     PATIENTS.TWO(KEEP=DIAG1-DIAG60)
4     PATIENTS.LAST(KEEP=DIAG1-DIAG60);
5  ARRAY DX {*} ALL;      *** retain all diagnoses/patient;
6  DO I=1 TO DIM(DX);
7  IF DX(I) > ' ' THEN DO;    CODE=DX(I); OUTPUT; END;
8  END;
9  PROC SORT NODUP; BY CODE;  *** sort - eliminating duplicates;
10 DATA OURDX;    MERGE DICTNARY DXCHECK(IN=INOURS); BY CODE;
11 FILE DXOUT;
12 IF _N_=1 THEN PUT  '/USERID JOB FMTDX,etc...'
13    '/EXEC SAS,REGION=3500K'
14    '/SASLIB DD DSN=USERID.SAS.FORMATS,DISP=OLD'
15    '/PROC FORMAT LIBRARY=SASLIB; VALUE $DIAGNOS';
16 IF INOURS;  *** keep only diagnoses actually seen;
17 PUT @COL "" @COL+1 CODE @COL+7 111=11'
18 @COL+10 DIAGDEF @COL+34 "" @;
19 COL+40;  *** output 2 definitions per line;
20 IF COL>80 THEN DO; PUT; COL=3; END;
21
Lines 2 & 4 - KEEPs in the SET statements bring into the data step only those variables specified. These data sets contain hundreds of variables but only diagnosis variables are needed.

Line 2 - The KEEP in the DATA statement retains only the variable specified. Data set OURDX will contain the single variable CODE. Without KEEP the original 60 variables would be retained and the SORT would be outrageously timeconsuming.

Lines 5 and 6 - the special SAS parameters {*} and ALL in the ARRAY statement as well as the DIM function in the DO statement allow one to define an array whose number of variables can change from time to time. Although the array currently has 60 elements, governmental requirements could change this number at any time.

Line 7 checks all 60 diagnoses per patient and outputs one observation for each diagnosis that is not blank. Potentially the new data sets DXCHECK could have 60 times the number of observations in the input sets. All the patients are not that ill so only 5 to 10 observations are created per patient. This is still a large number of observations but keep in mind that the data set being created contains only a single variable.

Line 9 - The PROC SORT NODUP eliminates thousands of duplicate diagnoses: measles will appear only once.

Line 11 causes subsequent PUTs to output to PDS member associated with FILE.

Line 19 - the workhorse of the program - creates nearly 4000 definitions for the PROC FORMAT statement. Figure 6 gives the results. For compactness, two elements of the code are output per line.

Without the DQUOTE option, line 19 would be:

```
PUT @COL "" @COL+1 DX @COL+7 ""="" @COL+10 DIAGDEF @COL+34 "" @;
```

The trailing @ permits outputting to the same line by a succeeding PUT. Line 21 checks for the end-of-line condition. The PUT with no operand outputs the filled line and COL is reinitialized to begin outputting on the following line.

Line 20 - the sum statement controls the COL pointer which rocks from COL 3 to COL 43. A sum statement automatically retains a value from observation to observation making the RETAIN statement at line 3 unnecessary. RETAIN is an easy way to initialize the accumulator variable COL.

891
The program in Figure 5 creates this SAS job:

```sas
//USERID1 JOB FMTDX,etc...
// EXEC SAS,REGION=3500K
//SASLIB DD DSN=USERID1.SAS.FORMATS, DISP=OLD
PROC FORMAT LIBRARY=SASLIB; VALUE $DIAGNOS
  'E804.8'='FALL FROM TRAIN-PERS NEC'
  'E805.2'='HIT BY TRAIN-PEDSTRIAN'
  'E810.0'='MV-TRAIN COLL-DRIVER'
  'E810.1'='MV-TRAIN COLL-PASNGR'
  'E993.'='WAR INJURY:EXPLOSION NEC'
  'V02.6'='VIRAL HEPATITIS CARRIER'
  'V10.72'='HX-HODGKINS DISEASE'
  'V10.79'='HX-LYMPHATIC MALIGNANT'
  '003.1'='SALMONELLA SEPTICEMIA'
  '004.0'='SHIGELLA DYSENTERIAE'
  '011.40'='TB LUNG FIBROSIS-UNSPEC'
  '035.'='ERYSIPelas'
;
```

**ADVANTAGES**

* Elimination of many expensive sorts
* Shorter code
* Availability to other users in format library

**CONCLUSIONS**

* The single FORMAT statement in Figure 1 is much easier than multiple sorts and merges with their associated costs.
* Although there is sufficient space to include 15,000 definitions, pruning unneeded ones is cost-saving. In addition, checking for here-tofore undefined diagnoses is helpful, too.
* A permanent format library is extremely valuable for one's own use as well as to make available to colleagues with similar needs.

Author's Address:

Judith B. Gabor
Quality Assurance - Trailer 71
University of California Davis Medical Center
2315 Stockton Boulevard
Sacramento, CA 95817

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

892