INTRODUCTION

The SASware Ballot(R) is the vehicle for SAS(R) users to identify additional features they would like to see in the SAS system. Response to the Ballot over the past few years indicates that many SAS features are not being used to their full potential. Several recurring requests for new features involve problems that can already be solved using existing techniques. The priorities for new features seem to be heavily geared toward the microcomputer products and full screen facilities. This paper is designed to deal with some basic base SAS items and to emphasize that there is no substitute for "trying" and experimenting with what's already around.

The following discussion will provide available solutions to some of the most frequent requests, focusing on data management, formatting, table look-up and recoding, indexing or direct access by key into SAS data sets, and methods for reading data dictionaries or codebooks. The examples demonstrate the base product's greater capabilities and suggest imaginative approaches to programming in SAS software.

I. THE MOST FREQUENTLY OCCURRING REQUESTS

In examining the last 5 Ballots, it is apparent that PROC FORMAT used in conjunction with the PUT function will provide the solution to the most frequently occurring requests, notably:

- Recoding facility
- Table look-up
- Shortening of list comparisons
- Storing of formatted values.

When we think of recoding, we generally think of a series of IF statements that test the values of coded variables, and assign recoded/translated values to a new variable:

```
DATA NEW;
SET SASDATA.CHIPS;
LENGTH TYPE $'8;
IF PRODUCT = '"' THEN TYPE = 'CHOC CHIP';
ELSE IF PRODUCT = '2' THEN TYPE = 'CHOC CHOC CHIP';
ELSE IF ( etc., etc., )
```

Recoding can also be accomplished with the PUT function. The PUT function is used to specify an output format for a value. The trick is to realize that the resulting format can be stored as a new variable, as well:

```
PROC FORMAT;
VALUE $TYP
  1 = 'CHOC CHIP'
  2 = 'CHOC CHOC CHIP'
  3 = 'PECAN'
  4 = 'MACADAMIA'
  5 = 'PEANUT BUTTER CHIP';
DATA NEW;
LENGTH COOKTYP $'8;
SET SASDATA.CHIPS;
COOKTYP = PUT(PRODUCT, $TYP.);
```

The resulting variable can be saved in the new SAS data set. A LENGTH statement should be coded to pre-set the length to accommodate the longest formatted value. The formatted value need not be stored, however, since the format library can be stored more efficiently than the additional, potentially long, new character variable.

The same technique can be used for table look-up:

```
DATA SUBSET;
SET SASDATA.CHIPS;
IF PUT(PRODUCT, $TYP.) = 'PECAN';
```

In this case, a new variable is not created, but the formatted value of the variable PRODUCT is used in the subsetting IF statement. The "table" is in the form of a VALUE statement within a PROC FORMAT. The translated value is then tested:

```
PROC FORMAT;
VALUE $DIV
  001 = 'YES'
  002 = 'NO'
  003 = 'NO'
  004 = 'YES'
  101 = 'NO'
  102 = 'YES'
  220 = 'YES'
  221 = 'NO';
DATA SUBSET;
SET SASDATA.CHIPS;
IF PUT(DIVISION, $DIV.) = 'YES';
```
Users are also interested in the ability to shorten list comparisons, as: IF X = 1 or X ~ 6 or X = 7 THEN ... The solution is closely related to the above topics:

PROC FORMAT;
VALUE $REG1F 101, 102, 204, 202, 300 = '1'
VALUE $REG2F 601, 603, 607, 709, 711 = '1'
VALUE $REG4F 201, 312, 313, 462, 900 = '1'

DATA REGION2; * Select Region 2;
    SET SASDATA.CHIPS;
    IF PUT(DISTRICT, $REG2F. 1 '1';
    ** use Region 2 format;
Likewise, users need to SORT, MERGE, or UPDATE according to the formatted values of the BY variable. The solution requires that the character string resulting from the PUT function be stored in a new variable:

PROC SORT DATA = NEW;
    BY COOKYTYP;
DATA SUBSET;
    SET LIST;
    IF KEY < 100 or KEY > 200;
    DO I = F_KEY TO L_KEY;
    SET MAIN
    END;

III. DATA RETRIEVAL

Direct access into a SAS data set is an efficient data retrieval method and requires an indexing system. This can be accomplished using the POINT option on the SET statement in conjunction with an "inverted list" that has been created in parallel with the primary SAS data set to be accessed:

DATA MAIN (DROP=F_KEY L_KEY)
LIST (KEEP=KEY F_KEY L_KEY);
    SET OLD;
    BY KEY;
    LENGTH F_KEY L_KEY 3;
    RETAIN F_KEY L_KEY;
    IF FIRST.KEY THEN F_KEY = _N_;
    IF LAST.KEY THEN DO;
    L_KEY = _N_; OUTPUT LIST;
    END;
OUTPUT MAIN;

In the MAIN file, the observations list attributes (KEY) for each record. In the LIST file, the observations list one record (in the MAIN file) for each attribute; in other words, lists of observation numbers having a given value of the KEY.

Data retrieval is accomplished by first determining which KEY values are the subset criteria. The LIST file is then queried using the observation numbers retrieved from LIST:

DATA SUBSET;
    SET LIST;
    IF KEY = 100 or KEY = 200;
    DO I = F_KEY TO L_KEY;
    SET MAIN POINT = I;
    END;

III. READING DATA DICTIONARIES AND CODE BOOKS

A common need, among survey processors in particular, is the ability to read data dictionaries or codebooks and generate SAS code: INPUT statements, LENGTH and FORMAT statements, range edits, and even missing value conversions (99 to a SAS missing value, for example). The concept of a generalized "edit" program would involve the use of fixed format control files or data dictionaries with details on each variable:

1 IDNUM RECORD ID 1-4 A
2 1-9999 = VALID
1 CASECTL CASE CONTROL STATUS 5-7 N
2 0, 9 = VALID
1 IVYY YEAR OF INTERVIEW 8-9 N
2 75-83 = VALID
2 88 = MISSING A
2 99 = MISSING B

Information would include the SAS variable name, label, column location in the raw data, variable type (character, integer, real), valid values, and values that represent missing data.

Two programs would be required: 1) a program to read the hierarchical control file, then generate the desired SAS statements and route them to external files using PUT and FILE statements, and 2) a driver program to read the open code and to build a program that creates the SAS data and then edits it.

The concept of the generalized edit has merely been suggested in this paper. Further information, with expanded code and examples, is available from the authors.
IV. SOLUTIONS TO MORE USER REQUESTS

The following are some short programming and data step tips that solve more SASware Ballot requests:

- Default Formats for Numeric Variables. Use the _NUMERIC_ keyword in a format statement:

```sas
DATA TEST;
SET SASDATA.CHIPS;
FORMAT _NUMERIC_ 8.4
REVENUE DOLLAR11.2;
```

- Use FIRST., and LAST., or other SAS special variables, in an ARRAY, and assign the values to new variables:

```sas
DATA TEST (DROP=1);
SET SASDATA.CHIPS;
BY PRODUCT;
F_PROD = FIRST. PRODUCT;
L_PROD = LAST. PRODUCT;
ARRAY VARS [*] F_PROD L_PROD;
DO I = 1 TO DIM(VARS);
IF VARS [I] = 1 THEN OUTPUT;
END;
```

- Use an ARRAY and an iterative DO loop to set all numeric missing values to zero or another specified value:

```sas
DATA TEST;
SET SASDATA.CHIPS;
ARRAY VARS [*] _NUMERIC_;
DO I = 1 TO DIM(VARS);
IF VARS [I] = . THEN VARS [I] = 0;
END;
DROP I;
```

- Use an ARRAY with an iterative DO loop to simulate a "FUZZ" option to round all numeric variables to a certain precision:

```sas
%MACRO RNDALL;
ARRAY NUM [*] _NUMERIC_;
DO I = 1 TO DIM(NUM);
NUM [I] = ROUND(NUM [I], .01);
END;
DROP I;
%MEND RNDALL;
```

- Error Message Manual. An invocation of PROC SASMSG can provide an error message manual, listing all numbered errors (including MACRO) with the associated explanation.

- More Applications Guides. The SUGI Proceedings provide an excellent reference manual dealing almost exclusively with applications.

- Simplified Output with BY/DO Statements. To avoid duplicate values of the ID variable from being printed on lines below the first occurrence, use both the ID and BY statements with the PROC PRINT, specifying the same variable on each one.

- Testing for Null Datasets. To check to see if a SAS dataset is empty prior to using the dataset, a generalized SAS Macro could be developed:

```sas
%MACRO EMPTYS (IN=);
DATA _NULL_
  IF _N_ = 1 AND LASTREC THEN
    CALL SYMPUT ('EMPTY', 'YES');
    SET &IN END = LASTREC;
  STOP;
%MEND EMPTYS;
```

The SYMPUT function will assign a value of 'YES' to the Macro variable EMPTY when the specified input dataset has no observations. The Macro variable EMPTY may then be checked using %IF logic to control further processing. Generalized SAS Macros such as this may be stored in a Macro utility library for use in many applications.

- Reducing the size of a SAS dataset:

  - For bit flags with 0,1 values, use character variables with a length of 1.
  - Use the LENGTH statement to minimize storage requirements for INTEGER numeric data values (see SAS User's Guide, p. 157).
  - Be careful when using character handling functions, noting the lengths of the variables returned. Be aware of the difference between built-in functions (where the SAS supervisor can determine the length at compile time) and library functions (that always return a length of 200). Use the LENGTH statement for resulting variables.
  - Store coded values in the SAS dataset and store formats separately, recoding only when necessary.
  - Minimize the amount of historical information stored: (refer to GEN = option).
References


The authors may be contacted at:
ORI, Inc.
Software Applications and Training Division
122 C Street, N.W., Suite 250
Washington, D.C. 20001
(202)737-2666