SAS® Macro Application Development: Automating the Data Step and INPUT Processes

S. Patrick Thornton, University of California San Francisco, San Francisco, CA

ABSTRACT

This paper discusses and outlines the application development process used by the author along with the goals of modularity, reusability, and scalability. The development and implementation of the Automated Input Interface (AII) serves as an illustration. The AII was created as a tool for the intermediate base SAS software user in order to facilitate batch processing of raw data through the specification of a simple set of parameters, rather than through the direct creation of syntax (Thornton, 1999). A meta-data set determines the data set names, variable names, and variable types for the conversion of raw data sets to SAS System data sets. A macro variable naming technique to simulate multidimensional arrays, the use of CALL SYMPUT and CALL EXECUTE are demonstrated as techniques crucial to the implementation of the AII.

INTRODUCTION

The introduction describes the major considerations of the author in developing the AII including discussion of the context, functional requirements, analysis of redundancy, and creating modularity. The other sections of the paper discuss specific macro programming techniques that were key to this application's implementation.

CONTEXT

The development of the AII began with the recognition of certain key features in the context of program development including administrative support for information management, future processing requirements, and past programming strategies.

- At this location, only base SAS software for batch mode was licensed and supported. Job Control Language was also required for program execution on an MVS operating system.
- Administration desired greater information processing speed, capacity, and sophistication but they were unwilling and unable to institute a change of technological infrastructure.
- Production reports were all prepared as ad hoc reports with redundant functionality and little compatibility in source code or format
- Ad hoc information requests accounted for a substantial portion of staff programming time

FUNCTIONAL REQUIREMENTS

From the previous assessment solutions were needed to:

- Leverage the existing IT
- Provide the basis for a consolidated production report process
- Allow for increased information generation while reducing day-to-day programming complexity
- Reduce percent of FTE devoted to ad hoc requests for information
- Smooth the technical transition from ad hoc information gathering to production reporting

IDENTIFYING REDUNDANCY

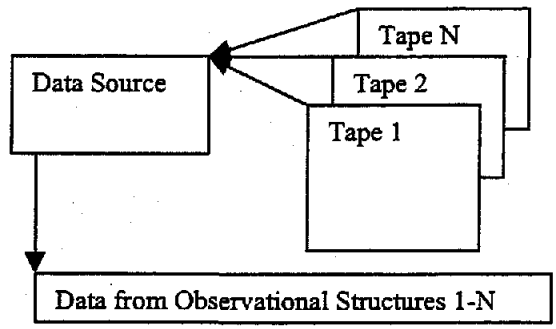
With these goals in mind, the author evaluated the capabilities of base SAS Software System and the macro facility (SAS Institute, Inc. 1997) and reviewed previously coded SAS system programs. The immediate goal was to identify and organize redundancies in the programs to determine which tasks might better be performed with centralized SAS macro programs. Effective modularized code allows centralized modification and error handling, as well as reusability and scalability. Redundancies were identified in the structure, process, and function of the production programs.

STRUCTURAL REDUNDANCY

A major source of redundancy across SAS System programs was produced from the structural similarity of the data sources used for production reports. All major data sources had the following characteristics (Figure 1):

- Tape data were sequentially stacked VSAM records
- Each structural data source was stored across multiple tapes (Figure 1)
- Data from each data source could belong to one of several observational structures or types

FIGURE 1. TAPE, DATA SOURCE, AND OBSERVATION TYPES



These characteristics in turn dictated some common aspects of the SAS System program structure that had been used in the past (Figure 2):

- Separate and identical SAS data steps were required to access each tape (Figure 2 *)
- Like SAS data sets from the same data source, but different tapes, were appended (e.g. A1-n from Tape 1-n)
- Most INPUT and all OUTPUT statements producing data sets had to be conditional on the value of two variables (e.g. R1, S1) common to all observations
- Most programs accessed multiple data sources

FIGURE 2. ILLUSTRATION OF REDUNDANCY IN DATA STEP

```
Data A1 (keep = R1 S1 A1) B1 (keep=R1 S1 B1);
        Infile tape 1 – n*;
        Input R1 S1 @;
        If R1 = "A" then do;
                If S1 = "1" then do;
                        Input A1;
                        Output A1;
                End;
        End;
        If R1 = "B" then do;
                If S1 = "0" then do;
                        Input B1;
                        Output B1;
                End;
        End;
Run;
```

*Indicates the whole data step is repeated to access tapes 1-n in order to acquire data from a complete data source

PROCESS REDUNDANCY

A logical but redundant input process was identified. Creating SAS system data sets with homogeneous observations from a particular data source always involved a specific order of operations: (a) creating data sets from each tape 1-n where the data steps were identical; (b) using an unconditional INPUT statement in each data step to access data common to all observations (e.g. R1, S1); (c) using INPUT statements conditional on the values of the 2 common variables to create variables from the data that were particular to the observation type (e.g. A1, B1); (d) using OUTPUT statements to separate heterogeneous observations into homogeneous data sets (e.g. A1, B1 in Figure 2); (e) appending all SAS data sets of like observational structure created from different tapes.

FUNCTIONAL REDUNDANCY

The most important functional redundancy across production report programs involved the accessing of data sources. It was found that a data source may be accessed as many as 6 times in one report process by programs that each used highly overlapping subsets of variables and observations. The ideal was to access only once, a superset of variables and observations that all existing programs could utilize, thereby creating standardization and saving resources. As an example, Figure 3 shows the number of times each program in a production report process accessed a data source to acquire data of a particular observation type. The marginal column totals provide a measure of undesirable redundancy in access to data of each observation type. Since accessing any observation type in a data source required exclusive tape access then the sequential execution of the programs in the example created 4 exclusive tape accesses rather than 1. Further, since data sources were located across many tapes exclusive access increased multiplicatively (e.g. 4 tape accesses/per data source when data source was located on 5 tapes equaled 20 exclusive tape accesses).

In this example, since data of each observation type was needed by at least one program, the superset of observations needed to accommodate all programs included data from AA, JJ, and CL observation types. A similar analysis was done to identify the superset of variables needed from each data source and observation type in order to accommodate all programs.

FIGURE 3. DESCRIBING ACCESS REDUNDANCY

Data Sources	CM			Total
Program Type	AA	JJ	CL	
Program1	1			1
Program2		1	1	2
Program3	1		1	2
Program4			1	1
Total	2	1	3	6

CODE REDUNDANCY TO MODULAR CONSISTENCY

The AII was designed to reduce the complexity of SAS data set creation by modularizing redundant code, modeling redundant processing, reducing functional redundancy, and shielding the programmer from most of the details. The AII accomplishes the task in 3 parts:

- A meta-data set of information necessary to create INPUT statement syntax and the data steps
- An interface to collect user specified values that determine the data sources to access, the number of tapes to access, the observations to input, and the variables to create
- SAS macro programs that use the meta-data set and the user-supplied parameters to build the necessary SAS system syntax

The AII meta-data set and the interface were the subject of an earlier paper (Thornton, 1999), so each will only be briefly discussed. Figure 4 shows an example of a meta-data set that contains the following variables: data source (tap), record (recor), sub-record (subrec), variable name (varn), variable location (varl), format indicator (c), data access format (daf), and variable role (reci). The interface allows you to specify the path(s) to the data source, as well as the number of physical locations from which to acquire data, the data from various observation types to include, (each to be stored as a homogeneous data set) and the variables to include in each data set (Figure 4).

Several SAS macro programs were used to translate the meta data set and user input shown in Figure 4 to the SAS System syntax shown in Figure 5. The macro programs allow considerable flexibility in the creation of data sets. This paper will discuss SAS macro modularization and several of the most important techniques used in the application programs.

FIGURE 4. AII: EXAMPLE CONFIGURATION

```
*************************************************************************;
*Locations of raw data;
%LET CM1 = C:\PROJECTS\SAS PRESENTATION\CM1.TXT;
%LET CM2 = C:\PROJECTS\SAS PRESENTATION\CM2.TXT;
*************************************************************************;
*Meta data set used to standardize SAS data set creation;
DATA TEMP2;
  INPUT TAP $ RECOR $ SUBREC $ VARN $ VARL $ C  DAF $
reci $;
  IF DAF = 'H' THEN DAF = '';
  CARDS;
  CM AB 0 spn          23-30    1        H    H
  CM AB 0 CDI          2-4      1        H    P
  CM AB 0 CAS          5-16     1        H    P
  CM AB 0 OFF          $31-40   1        H    H
  CM AB 0 OFFDAT       @41      2 MMDDYY6.    H
  CM AB 0 RTY          $17-18   1        H    R
  CM AB 0 SNU          19-21    1        H    P
  CM AB 0 CCN          22       1        H    S
  CM LQ 1 CDI          2-4      1        H    P
  CM LQ 1 CAS          5-16     1        H    P
  CM LQ 1 RTY          $17-18   1        H    R
  CM LQ 1 SNU          19-21    1        H    P
  CM LQ 1 CCN          22       1        H    S
  CM LQ 1 POI          $23-24   1        H    H
  CM LQ 1 POIDAT       @33      2 MMDDYY6.    H
  ;
*************************************************************************;
*User Interface to allow selection of each data source where each
has the following components;
* physical location (t);
* number of physical locations (NT);
* number of records (NR) from the physical location;
* records and sub-records (R1..R10) of data at physical location;
DATA FIRST;
  INPUT T $ NT NR R1 $ R2 $ R3 $ R4 $ R5 $ R6 $ R7 $ R8 $
R9 $ R10 $;
  RETAIN K 0;
  K = K + 1;
  CALL SYMPUT('DS',K);
  DROP K;
  CARDS;
 CM 2 2 AB0 LQ1 H H H H H H H H
  ;
*************************************************************************;
*Choose the specific variables from a data source, record, and
sub record to include in the input statements created by the
program;
```

```
*As specified in data set first set the following parameters:
*  the physical location (TAP);
*  the record (RECOR) from the physical location;
*  the sub record (SUBREC) from the physical location;
*  the variable literal names (V1..V15) and place holders (H);

DATA SEC;
 INPUT TAP $ RECOR $ SUBREC $ V1 $ V2 $ V3 $ V4 $ V5 $
V6 $ V7 $ V8 $ V9 $ V10 $ V11 $ V12 $ V13 $ V14 $ V15 $;
 CARDS;
 CM AB 0 SPN CDI OFF H H H H H H H H H H H H
 CM LQ 1 CAS POI POIDAT H H H H H H H H H H H H
 ;
*************************************************************************
;
%ALLDS(&DS, BOGUS);
```

FIGURE 5. All: EXAMPLE LOG

```
MPRINT(ALLTAP):  DATA CMAB01 (KEEP
MPRINT(ALLTAP):  = SPN CDI CAS OFF OFFDAT RTY SNU
CCN ) CMLQ11 (KEEP
MPRINT(ALLTAP):  = CDI CAS RTY SNU CCN POI POIDAT );
MPRINT(ALLTAP):  INFILE "C:\PROJECTS\SAS
PRESENTATION\CM1.TXT";
MPRINT(ALLTAP):  INPUT
MPRINT(PUTALL):  CDI 2-4 CAS 5-16 RTY $17-18 SNU 19-21
CCN 22
MPRINT(ALLTAP):  @;
MPRINT(ALLTAP):  *IF _N_ > 40000 THEN STOP;
MPRINT(PUTALL):  IF RTY = "AB" THEN DO;
MPRINT(ALLTAP):  ;
MPRINT(PUTALL):  IF CCN = 0 THEN DO;
MPRINT(ALLTAP):  ;
MPRINT(ALLTAP):  INPUT
MPRINT(PUTALL):  SPN 23-30 OFF $31-40 @41 OFFDAT
MMDDYY6.
MPRINT(ALLTAP):  ;
MPRINT(ALLTAP):  OUTPUT CMAB01 ;
MPRINT(ALLTAP):  END;
MPRINT(ALLTAP):  END;
MPRINT(PUTALL):  IF RTY = "LQ" THEN DO;
MPRINT(ALLTAP):  ;
MPRINT(PUTALL):  IF CCN = 1 THEN DO;
MPRINT(ALLTAP):  ;
MPRINT(ALLTAP):  INPUT
MPRINT(PUTALL):  POI $23-24 @33 POIDAT MMDDYY6.
MPRINT(ALLTAP):  ;
MPRINT(ALLTAP):  OUTPUT CMLQ11 ;
MPRINT(ALLTAP):  END;
MPRINT(ALLTAP):  END;
MPRINT(ALLTAP):  RUN;

NOTE: The infile "C:\PROJECTS\SAS
PRESENTATION\CM1.TXT" is:
    FILENAME=C:\PROJECTS\SAS PRESENTATION\CM1.TXT,
    RECFM=V,LRECL=256

NOTE: 10 records were read from the infile "C:\PROJECTS\SAS
PRESENTATION\CM1.TXT".
    The minimum record length was 0.
    The maximum record length was 47.
NOTE: SAS went to a new line when INPUT statement reached
past the end of a line.
NOTE: The data set WORK.CMAB01 has 9 observations and 8
variables.
NOTE: The data set WORK.CMLQ11 has 0 observations and 7
variables.
NOTE: The DATA statement used 0.38 seconds.

MPRINT(ALLTAP):  DATA CMAB02 (KEEP
MPRINT(ALLTAP):  = SPN CDI CAS OFF OFFDAT RTY SNU
CCN ) CMLQ12 (KEEP
MPRINT(ALLTAP):  = CDI CAS RTY SNU CCN POI POIDAT );
MPRINT(ALLTAP):  INFILE "C:\PROJECTS\SAS
PRESENTATION\CM2.TXT";
MPRINT(ALLTAP):  INPUT
MPRINT(PUTALL):  CDI 2-4 CAS 5-16 RTY $17-18 SNU 19-21
CCN 22
MPRINT(ALLTAP):  @;
MPRINT(ALLTAP):  *IF _N_ > 40000 THEN STOP;
MPRINT(PUTALL):  IF RTY = "AB" THEN DO;
MPRINT(ALLTAP):  ;
MPRINT(PUTALL):  IF CCN = 0 THEN DO;
MPRINT(ALLTAP):  ;
MPRINT(ALLTAP):  INPUT
MPRINT(PUTALL):  SPN 23-30 OFF $31-40 @41 OFFDAT
MMDDYY6.
MPRINT(ALLTAP):  ;
MPRINT(ALLTAP):  OUTPUT CMAB02 ;
MPRINT(ALLTAP):  END;
MPRINT(ALLTAP):  END;
MPRINT(PUTALL):  IF RTY = "LQ" THEN DO;
MPRINT(ALLTAP):  ;
MPRINT(PUTALL):  IF CCN = 1 THEN DO;
MPRINT(ALLTAP):  ;
MPRINT(ALLTAP):  INPUT
MPRINT(PUTALL):  POI $23-24 @33 POIDAT MMDDYY6.
MPRINT(ALLTAP):  ;
MPRINT(ALLTAP):  OUTPUT CMLQ12 ;
MPRINT(ALLTAP):  END;
MPRINT(ALLTAP):  END;
MPRINT(ALLTAP):  RUN;

NOTE: The infile "C:\PROJECTS\SAS
PRESENTATION\CM2.TXT" is:
    FILENAME=C:\PROJECTS\SAS PRESENTATION\CM2.TXT,
    RECFM=V,LRECL=256
```

MACRO APPLICATION IMPLEMENTATION

As a general overview, the macro programs generally had to conduct operations for each data source. For each data source, the programs had to build a data step accessing each tape source across which the data source was stored. For a given data source and tape source then, the programs had to create a single data step that produced SAS data sets 1 to n of the user's choosing and that contained user selected variables from the meta data. The complete programs will not be presented, however selected techniques will be discussed with accompanying code.

The meta data set (TEMP2) and user-specified parameters in the data sets FIRST and SEC, as well as various macro variables supplied input to the macro programs. The type and form of the user specified values in the interface was created with brevity and past nomenclature in mind in order to ensure some familiarity. A basic task in the macro application was to save the meta data set information to macro variables, thereby making it available for the duration of program execution. The availability of the meta data was crucial to creating data steps that dynamically included that information as syntax.

SIMULATING MULTIDIMENTIONAL ARRAYS

The CALL SYMPUT statement was used as the basis to save the information in the meta data set to macro variables. CALL SYMPUT allows macro variables to be created that change

dynamically in name and resolution with the observations of a data set. Since the observations of several data sets in the interface program were used as macro program input, the functionality of CALL SYMPUT was ideal. However, a matrix of meta data information for a given data source had to be stored and systematically indexed in order to efficiently build each part of a data step's syntax. For example, TEMP2 contains 8 observations with several variables storing meta data about all variables found in the data source CM of observation type AB-0. In order to incorporate the meta information as syntax at the appropriate points in a data step a method of indexing the matrix was desirable. By concatenating the following, a naming convention was created.

- Data source (value of tap)
- Observation type (value of recor and subrec)
- Information type (defined the type of information stored)
- Index (number identifying the associated variable).

Figure 6 shows part of a SAS log file for a program that simulates multidimensional arrays using this naming convention. The first data step in the example saves the values of the variable VARN & VARL to macro variables with the use of CALL SYMPUT. The first argument to CALL SYMPUT is a concatenation of the values of TAP, RECOR, SUBREC, the macro variable P or P1, and K. P and P1 designate the information being stored, i.e. the values of VARN or VARL that are the second arguments to each CALL SYMPUT respectively. The variable k is a sub count. K counts the number of observations in the meta data set within TAP, RECOR, and SUBREC. In effect K is a unique index of meta information stored for each data source and observation type. The meta data is stored such that for any data source and observation type there is a simulated matrix of dimensions k x P. As a demonstration of this consider the second data step in Figure 6. The macro variables P & P1 are used as critical arguments to CALL EXECUTE statements involving the macro program ALLTAP. P and P1 resolve to N and M, thus critically determining which information results from the %put in ALLTAP. The second data step also uses the SAS observation loop and the variable T to form the macro variable NT where T varies from 1 to the number of unique TAP, RECOR, and SUBREC combinations found in the data set. N1-Nt macro variables were created with CALL SYMPUT in the first data step to record the total observations of meta data for each data source and observation, that is Nt stores the value of K. So, ALLTAP receives information on the data source (TAPE), the observation type (Rec,Sub), number of variables (NV), and information type (P) through the CALL EXECUTE statements each time the observation loop encounters the first observation of a TAP, RECOR, SUBREC subgroup. ALLTAP performs the trivial task of resolving the macro variables to place both the names and the values in the SAS log file. While this task is simple it is an example of a modularized, reusable, and scalable functionality.

FIGURE 6. ILLUSTRATION OF FUNCTIONALITY IN MACRO NAMING CONVENTION

```
127 %macro ALLTAP(Tape,rec,sub,nv,p);
128     %do I = 1 %to &nv;
129         %let vari = &tape&rec&sub&p&i;
130         %put &vari : &&&vari;
131     %end;
132 %mend;
133
134 PROC SORT DATA=TEMP2;
135     BY TAP RECOR SUBREC;
136 DATA TEMP2 (drop=t k);
137     SET TEMP2;
138     BY TAP RECOR SUBREC;
139     %let p = n;
140     %let p1 = m;
141     retain k 0;
142     RETAIN T 1;
143     K = k + 1;
144     CALL SYMPUT(LEFT(TRIM(Tap))||
LEFT(TRIM(recor))|| LEFT(TRIM(subrec))||"&p"||
LEFT(k), TRIM(varn));
145     CALL SYMPUT(LEFT(TRIM(Tap))||
LEFT(TRIM(recor))|| LEFT(TRIM(subrec))||"&p1"||
LEFT(k), TRIM(varl));
146     CALL SYMPUT("N"||LEFT(T),LEFT(K));
147     IF LAST.SUBREC THEN DO;
148         K = 0;
149         T = T + 1;
150         CALL SYMPUT("DS",LEFT(T));
151     END;
152 run;
153 DATA TEMP2;
154     SET TEMP2;
155     BY TAP RECOR SUBREC;
156     retain t 1;
157     IF FIRST.SUBREC THEN do;
158     s = symget("n"||left(t));
159     CALL
EXECUTE('%ALLTAP('||TAP||','||RECOR||','||SUBREC||','||s||','||"&
p"||')');
160     CALL
EXECUTE('%ALLTAP('||TAP||','||RECOR||','||SUBREC||','||s||','||"&
p1"||')');
161     t = t + 1;
162     end;
163 run;
CMAB0n1 : spn
CMAB0n2 : CDI
CMAB0n3 : CAS
CMAB0n4 : OFF
CMAB0n5 : OFFDAT
CMAB0n6 : RTY
CMAB0n7 : SNU
CMAB0n8 : CCN
CMAB0m1 : 23-30
CMAB0m2 : 2-4
CMAB0m3 : 5-16
CMAB0m4 : $31-40
CMAB0m5 : @41
CMAB0m6 : $17-18
CMAB0m7 : 19-21
CMAB0m8 : 22
CMLQ1n1 : CDI
CMLQ1n2 : CAS
CMLQ1n3 : RTY
CMLQ1n4 : SNU
CMLQ1n5 : CCN
CMLQ1n6 : POI
CMLQ1n7 : POIDAT
CMLQ1m1 : 2-4
CMLQ1m2 : 5-16
CMLQ1m3 : $17-18
CMLQ1m4 : 19-21
CMLQ1m5 : 22
CMLQ1m6 : $23-24
CMLQ1m7 : @33
PMAZ1n1 : SPN
PMAZ1n2 : NAME
PMAZ1n3 : GENDER
PMAZ1m1 : 1-10
PMAZ1m2 : $11-20
PMAZ1m3 : 21-22
```

It is further useful to let the value of p define the function that the data therein will serve in your program or the format within which that data will be incorporated. For example, creating a data step of the type in Figure 4 that incorporates the meta data as syntax required several passes through the macro naming array that is associated with each data source where the data is used in different ways. In fact a pass through the macro variable array was used to build the KEEP statements at the beginning of each data step, the first INPUT statements, both IF statements, and the conditional INPUT statements within the IF blocks. The function of placing the meta data correctly was modularized to the macro program PUTALL. The program was used to place the meta data in 4 out of 5 of these instances just listed by using the macro naming convention to simulate the multidimensional arrays, which allowed the information to be stored and indexed according to its function.

CONCLUSION

This paper has discussed some of the considerations made in application development and has illustrated several SAS macro techniques crucial to implementation of modular, reusable, and scalable application programs. While there may be many ways to implement the similar application's functions, it is hoped that the approach used here will aid others in the task of applications development and implementation.

REFERENCES

SAS Institute Inc.(1997). SAS macro language: Reference, First Edition. Cary, NC: SAS Institute Inc.

Thornton, S. P. (April, 1999). Automated input interface for SAS batch-mode processing. Paper presented and published in Coders Corner at the SAS Users Group International Conference, Miami, Florida.

TRADEMARKS

SAS software is a registered trademark of trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

ACKNOWLEDGEMENTS

Thanks to Sarah M. Barry of Child Services Research Group for editing.

CONTACT INFORMATION

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

S. Patrick Thornton
Child Services Research Group
44 Montgomery Street Suite 1450
San Francisco, CA 94104-4602
(415) 502-8004
SPThornton1@juno.com