Introduction

Many data services departments offer their open-shop users a wide variety of analytical processing tools. This creates problems for users who must use separate languages or environments and then exchange data between systems. SAS, being one of the most complete and heavily used of these tools, made it the logical place to attempt centralizing much of the processing. This paper addresses one area of the centralization problem; namely accessing callable subroutines from within the DATA step.

Advantages of this implementation include: 1) object or load modules can be dynamically link-edited and then executed; 2) subroutines do not have to be rewritten to handle conversion of argument types; 3) any SAS user familiar with FORTRAN, PL/I, or Assembler can write their own subroutine and use it in a SAS DATA step; 4) writing a subroutine is often easier than either writing a SAS procedure or creating external data files to handle interfacing.

History of Problem

Many of our current SAS users were, at one time, using FORTRAN exclusively to solve their analytical, storage and retrieval, and reporting problems. Some vendor-written packages, such as IMSL (International Mathematical and Statistical Libraries, a library of FORTRAN-written subroutines) were purchased and a great deal of in-house user programming was done. As SAS usage spread throughout the Company, more and more of the FORTRAN programmers were converting systems to SAS. Eventually, SAS usage far outweighed the FORTRAN usage in the user community.

The first problem that we encountered was users wanting to access IMSL and user-written subroutines to do their analytical processing that SAS did not do, but still use SAS to do the data handling, reporting, and graphing. This first problem was solved, though not conveniently, by the user running a two-step job: One to execute a FORTRAN program which used IMSL to do the processing and then wrote the results to an external data set, and a second step to execute SAS, read the external data set, create a SAS data set, and print the results. (Later, we realized that this process could have been simplified somewhat by executing the FORTRAN program via a SAS PROC statement.)

The second problem was similar to the first, except that the users wanted to access the IMSL and user-written subroutines dynamically within the SAS DATA step. The solution to this second problem was to use the CALL statement (which, happily, is now documented as a statement in SAS Users Guide: Basics). The IMSL and user-written subroutines were rewritten so that all of the arguments were double-precision reals, the default for a numerical variable or constant in SAS. The library containing the subroutine load module was included in the STEPLIB DD statement in the SAS JCL procedure; the user was then able to call his subroutine from within a SAS DATA step.

Several difficulties arose with the solution to the second problem:

1. Every subroutine to be used in SAS had to be rewritten. Arguments which were integer (what SAS calls integer binary) had to be changed to real (what SAS calls floating point or real binary). Users were unhappy not being able to use a subroutine immediately as the need arose, but first had to go through the step of rewriting it.

2. Users were also having to maintain two sets of user-written subroutines - one for the FORTRAN application, where integer variable types were used, and another set for use in SAS. Additionally, IMSL subroutines which were rewritten had to be kept track of by the users.

3. A concern of the Data Services Department was the additional amount of disk space that was being used. Duplicate copies of both user-written and IMSL subroutines were being stored. The additional IMSL subroutines were especially a problem, since they were being stored as link-edited load modules instead of object modules and, therefore, used considerably more space. Also, leaving the IMSL source on tape was not satisfactory since it often took too long to run a tape job just to retrieve a subroutine; but to put the IMSL source on disk took considerable (by our standards) disk space.

4. Fortunately, IMSL distributes the source code for its subroutines; thus, we were able to modify the subroutines. But this only works if source code is (a) available, (b) either simple enough or so well...
written that changes can be made leaving a subroutine that still works, and (c) written in a language that the user understands, which for us ruled out subroutines written in anything but FORTRAN.

Our Solution

The solution to our problems was (1) to provide our users with subroutines which would handle the conversion of integer and character arguments without the necessity of making any changes to the subroutine being executed, thus solving Problems 1, 2, 4, and part of 3 above, and (2) to provide subroutines by which the user could invoke the linkage editor from within the DATA step, thus solving the problem in 3 above of having to create a load module version of an IMSL subroutine before SAS was executed and the subroutine called.

The Subroutines

The following four subroutines were written for the SAS user to call from within the SAS DATA step using the CALL statement:

1) I2R (R, XLNARY, XLNVAR, I)

DESCRIPTION: This subroutine will convert an INTEGER*4 array to a REAL*4 or REAL*8 array.

ARGUMENT DESCRIPTIONS:
R : REAL*4 or REAL*8 array to contain converted values
XLNARY: Number of elements in I
XLNVAR: Length of each element of R (4 or 8)
I : INTEGER*4 array to be converted

2) R2I (R, XLNARY, XLNVAR, I)

DESCRIPTION: This subroutine will convert a REAL*4 or REAL*8 array to an INTEGER*4 array.

ARGUMENT DESCRIPTIONS:
R : REAL*4 or REAL*8 array to be converted
XLNARY: Number of elements in R
XLNVAR: Length of each element of R (4 or 8)
I : INTEGER*4 array containing converted R values

3) LINK (ISBADR,ITPADR,DARG1,...,DARG16)

DESCRIPTION: This subroutine is used to dynamically load and execute any program module in the STEPLIB data sets.

ARGUMENT DESCRIPTIONS:
ISBADR: Name of the subroutine to be executed. Must be a character variable or character constant.
ITPADR: Type mask. Must be character variable or character constant. The type mask has the following format: 'XXXXXXXXXXXXX' (16 characters max)

Where each "X" is either:
"I" for INTEGER*4 variable or character constant
"C" for character variable
"*" for REAL*4 variable or REAL*8 variable or REAL*4 array or REAL*8 array or INTEGER*4 array

and indicates what argument type is to be used in the corresponding argument of the subroutine to be executed.

For example, if the subroutine to be executed uses six arguments, the first of which is an integer variable, the second is a REAL*4 variable, the third is an integer array, the fourth is a REAL*8 array, the fifth is an integer variable, and the sixth is a character variable, then the type mask would look like this: 'I**IC'.

If the subroutine requires:

a. an integer variable then a SAS variable with a length of 8 (the default) must be passed to the subroutine. The subroutine will convert the variable to an INTEGER*4 before executing the subroutine and then convert it back to its original form when the subroutine is finished.

e.g. if "INT" is to be an INTEGER*4 value passed to the subroutine, the required SAS statements are:

TYPE='**I*';
CALL LINK (SUBNAM, TYPE, A, B, INT, C);

b. A REAL*4 variable then the variable must be given a length of 4 in the SAS program.
e.g.
LENGTH X 4;
CALL LINK (SUBNAM, TYPE, ..., X, ...);

c. A REAL*8 variable then the variable must be
given a length of 8 in
the SAS program. (That
is the default - no
length statement is
required.)

e.g.
CALL LINK (SUBNAM,
TYPE, ..., X, ...);

d. A REAL*4 array then the
array variables must be
given a length of 4 and
the argument passed from
SAS is the first varia-
ble of the SAS array.

e.g.
LENGTH X1-X100 4;
ARRAY X (I) X1-X100;
CALL LINK (SUBNAM, TYPE,
..., X1, ...);

or
LENGTH X1-X100 4;
ARRAY X (I) X1-X100;
I=1;
CALL LINK (SUBNAM, TYPE,
..., X, ...);

e. A REAL*8 array then the
array variables must
have a length of 8
(default) and the argu-
ment passed from SAS is
the first variable of
the SAS array.

e.g.
ARRAY X (I) X1-X100 4;
CALL LINK (SUBNAM, TYPE,
..., X1, ...);

or
ARRAY X (I) X1-X100 4;
I=1;
CALL LINK (SUBNAM, TYPE,
..., X, ...);

f. An INTEGER*4 array then
an array of REAL*4 or
REAL*8 values in SAS
must first be converted
to INTEGER*4 values -
which may be done with a
call to R2I. The first
variable of the array
containing the integer
values is passed to the
subroutine.

e.g.
LENGTH X 4;
ARRAY X (I) X1-X100;
CALL R2I (X1,100,4,X1);
CALL LINK (SUBNAM, TYPE,
..., X1, ...);

or
LENGTH K 4;
ARRAY X (I) X1-X100;
ARRAY K (I) K1-K100;
CALL R2I (X1,100,8,K1);
CALL LINK (SUBNAM, TYPE,
..., K1, ...);

Note that the 1st and
4th arguments may or may
not point to the same
array; the real array
may have elements with
lengths 4 or 8, and the
integer array may have
elements with any length
but the total length of
the array may not be
less than (4 * number-
of-elements-in-the-
first-array-to-be-
converted). The first
example shown here uses
the least amount of
space, since the real
array is replaced by the
integer array.

g. A character variable or
can be a
SAS character constant
SAS character varia-
bale is used and the type
for that argument is set
to "C".

e.g.
NAME='SMITH';
TYPE='C***';
CALL LINK (SUBNAM, TYPE,
NAME, X, Y, Z);

or
CALL LINK (SUBNAM, TYPE,
'SMITH', X, Y, Z);

DARG1-DARG16: These are the argu-
ments to be used in the
subroutine to be executed,
in the order in which they
appear in the loaded sub-
routine.

4) LKED (MODNAME)
DESCRIPTION: This subroutine will
link edit the module whose name is
passed to this subroutine.

ARGUMENT DESCRIPTIONS:
MODNAM: Name of module to be
link edited. May be
character variable or
constant.
Example Programs

**Statistical Analysis System**


NOTE: THE JOB V0004385 HAS BEEN RUN UNDER RELEASE 79.6 OF SAS AT PORTLAND GENERAL ELECTRIC COMPANY (00715).

NOTE: SAS OPTIONS SPECIFIED ARE:


1

* THIS EXAMPLE COMPUTES VALUES OF THE BINOMIAL DISTRIBUTION.
* NO ARRAYS ARE USED IN THIS EXAMPLE SO NO CALLS TO I2R OR R2I ARE MADE.

DATA; KEEP K PK PS; * K (INPUT) K NUMBER OF SUCCESSES;
* N (INPUT) NUMBER OF TRIALS;
* P (INPUT) PROBABILITY OF A SUCCESS ON A SINGLE TRIAL;
* PK (OUTPUT) PROBABILITY OF K OR LESS SUCCESSES;
* PS (OUTPUT) VALUES OF THE BINOMIAL DISTRIBUTION. THIS CALL STATEMENT INVOKES THE LINKAGE EDITOR TO LINK EDIT THE OBJECT MODULE "MDBIN";

N=20; P=0.75;
DO K = 0 TO N;
CALL LINK ('MDBIN', '11***1', K, N, P, PS, PK, IER);
END;

NOTE: THE VARIABLE PS IS UNINITIALIZED.
NOTE: THE VARIABLE PK IS UNINITIALIZED.
NOTE: DATA SET WORK.DATAI HAS 21 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.90 SECONDS AND 228K.

PROC PRINT;
NOTE: THE PROCEDURE PRINT USED 0.27 SECONDS AND 286K AND PRINTED PAGE 1.
NOTE: SAS USED 286K MEMORY.

NOTE: SAS INSTITUTE INC. SAS CIRCLE BOX 8000 CARY, N.C. 27511-8000

**Statistical Analysis System**

1 388
Example Programs (continued)

1 ST AT I STI CA L A NA L Y S I S S Y S TEM
9:45 MONDAY, JANUARY 10, 1983

NOTE: THE JOB V0004386 HAS BEEN RUN UNDER RELEASE 79.6 OF SAS
AT PORTLAND GENERAL ELECTRIC COMPANY (00715).

NOTE: SAS OPTIONS SPECIFIED ARE:
DEFAULT=LOCAL, LEAVE=25K, LS=102, SORT=1.

1 * THIS EXAMPLE GENERATES A RANDOM PERMUTATION OF THE
2 * INTEGERS 1 TO K
3 *
4 DATA;
5 ARRAY IPER1-IPER10;
6 SEED =1234567; * SEED USED FOR RANDOM PERMUTATION;
7 K=10; * PERMUTATION WILL BE FOR INTEGERS 1-10;
8 CALL LKED ('GGPER'); * LINK EDIT IMSL ROUTINE "GGPER";
9 CALL LINK ('GGPER', 'N', SEED, K, IPER1);
10 CALL I2R (IPER1, 10, 8, IPER1);
11 *
12 * THE SUBROUTINE GGPER RETURNED INTEGER RESULTS
13 * IN ARRAY IPER. I2R CONVERTS THESE INTEGER VALUES TO
14 * REAL*8 VALUES SO THEY CAN BE USED IN SAS.
15 *

NOTE: DATA SET WORK.DATAl HAS 1 OBSERVATIONS AND 12 VARIABLES. 190 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.47 SECONDS AND 256K.

16 PROC PRINT;
NOTE: THE PROCEDURE PRINT USED 0.27 SECONDS AND 302K AND PRINTED PAGE 1.
NOTE: SAS USED 302K MEMORY.

NOTE: SAS INSTITUTE INC.
SAS CIRCLE
BOX 8000
CARY, N.C. 27511-8000

STATISTICAL ANALYSIS SYSTEM
9:45 MONDAY, JANUARY 10, 1983

OBS IPER1 IPER2 IPER3 IPER4 IPER5 IPER6 IPER7 IPER8 IPER9 IPER10 SEED K
1 1 9 5 10 8 6 3 4 2 7 1985278849 10

* THIS EXAMPLE SEARCHES A VSAM DATABASE. GIVEN A USER'S LOGON ID,
* RETURN HIS NAME, DEPARTMENT NUMBER, AND TOTAL YEAR-TO-DATE
* CHARGES FOR HIS ACCOUNT.
*
DATA;
LENGTH YTDAMT 4;
NAME 8 20;
INFILE LOGON.MASTER;
INPUT LOGON_ID;
SUBNAM = 'LOGYTD';
TYPE = 'ICIN';
CALL LINK (SUBNAM, TYPE, LOGON_ID, NAME, DEPT, YTDAMT);
*
* NOTE THAT THE VARIABLE "NAME" HAD TO BE DEFINED AS A CHARACTER
* VARIABLE BEFORE THE CALL WAS MADE.
*
PROC PRINT;
Changes to the SAS JCL Procedure

Several changes were made to the SAS PROC:

1. SYSLIN, SYSLMOD, and SYSLIB DD statements were added to handle the link editing when the subroutine LKED is called. The SYSLIN data set is temporary and will contain linkage editor control statements. The SYSLMOD data set contains the link edited load module, and the SYSLIB data set is a concatenation of FORTRAN, PL/I, and PGE libraries often required for link editing.

2. In the concatenation of libraries in STEPLIB, an additional DD was added to refer back to the SYSLMOD data set. The SYSLMOD data set contains the link edited subroutine as output from subroutine LKED. When a CALL is made in a SAS DATA step, SAS searches the libraries in STEPLIB to find the module to be executed. This referback to SYSLMOD is similar to the way LIBRARY is referred back to in STEPLIB in order to access formats.

3. The library SAS.LOADLIB was added to STEPLIB and contains the subroutines I2R, R2I, LINK, LKED, and subroutines commonly called from within SAS.

4. An FTO6F001 DD statement was added. Many vendor and user subroutines write to Unit 6.

Conclusion

This paper has examined accessing vendor- and user-written subroutines from within a SAS DATA step using a CALL statement in conjunction with subroutines to handle integer variable and array conversions, character addressing, and subroutine link editing. This technique allows access to any subroutine without the need to make changes to the subroutine; thus the technique is especially appropriate for accessing vendor-written routines for which the source is not available.

Our experience has been that this technique allows our SAS users to continue to use many of the FORTRAN tools they require, still using SAS as the central system of analysis.

Anyone interested in obtaining copies of these subroutines, along with instructions for modifying the SAS JCL procedure appropriately can write to:

Richard A. Boro
Portland General Electric Co.
121 SW Salmon
Portland OR 97204