

Huixing Tang, Matthew Grover, Kerril Bauerly
The Psychological Corporation
Paper presented at the Second Annual South-Central
Regional SAS Users Group Conference, 1992

Techniques for Simultaneously Processing Multiple Data Sets with Different File Layouts Using SAS Flexcode

INTRODUCTION

It is not an uncommon situation for a data analyst to have data delivered for one project in a large number of files with similar but not identical record layouts. These files often have a different number of variables of the same type. The differences in the record layout and the varying number of variables make it impossible to concatenate the files and process them simultaneously in one data step. One way to deal with this problem is to write one version of SAS code and copy and change it for processing each file separately. Such a task is highly cumbersome, time-consuming and can lead to errors or inaccuracies. This is particularly true when a large number of files are involved and a lot of analyses are needed. In this type of situation, programming which can process multiple files simultaneously would be highly desirable.

The purpose of this paper is to explore how multiple files with different record layouts and different numbers of variables can be processed simultaneously using SAS flexcode: code that changes itself in the process of execution (Aster & Seidman, 1991). The second section of the paper discusses how to create a parameter file which provides the input data for writing flexcode. The third section presents two kinds of flexcode: (1) macro invocation statements created using the *call symput* routine or *put* statement, and (2) the open code generated using the *put* statement. The use of macro flexcode entails processing multiple files using one macro, while the use of open flexcode entails processing multiple files in a non-macro environment by first writing out open code and then executing it using the *%include* statement. These two approaches will be compared in efficiency in terms of CPU time, input/output operations, and memory storage. The fourth section discusses techniques to generate, within SAS, JCL and other mainframe program control language needed to make simultaneous processing possible with other programs.

The techniques discussed will be illustrated using an ongoing data processing application at The Psychological Corporation.¹ The application involves processing student responses to a mastery test currently being developed jointly by a state testing program and The Psychological Corporation. The test has many different forms covering a wide range of grade levels and content areas.

Each form has two types of items: (1) dichotomously scored multiple choice items and (2) polytomously-scored constructed-response items. The response data for each form were entered into a separate data file. In each data set, responses to the dichotomous items were entered first, followed by the responses to the polytomous items with or without a column space in between the two clusters. Not only may the number of items in each form vary, but the starting column position for each type of item differs from form to form. In our example, we will use 19 forms from the Reading Test and 31 forms from the Mathematics Test which share the aforementioned differences.

CREATING A PARAMETER FILE

The first step in the present approach is to create a parameter file: a file that contains all the necessary information or variables regarding the differences in the record layout. This file provides the basic input data for generating SAS flex code or other programming control language code. In the current application, the necessary information includes (1) the number of dichotomous items (DNUM), (2) the starting column position for the dichotomous items (DSTART), (3) the number of polytomous items (PNUM), and (4) the starting column position for the polytomous items (PSTART). Also important is the information for keeping track of each file and for reading in and writing out data such as (1) content area (STRAND), (2) grade/level (LEVEL), and (3) test form (FORM).

The parameter file can be created either as an external file or included in the program using the *cards* statement. The latter method is used here for illustrative purposes. The following SAS statements are used to create the parameter file:

¹ The Psychological Corporation is an internationally known test publisher.

```

data pfile;
    input strand $ 1 level 3-4 form 6-7 dnum 9-10
    dstart 12-13 pnum 15-16 pstart 18-19;
    cards;
r 6 51 10 7 7 18
r 6 52 10 7 5 16
. . . . .
m 10 69 30 9 4 40;

```

SIMULTANEOUS PROCESSING USING SAS

This section discusses methods of simultaneously processing multiple data files with the SAS data step and SAS procedures. The simultaneous approach can be accomplished with macros or with open code. Two methods (Method1 and Method2) of processing with macros and one method of processing with open code are discussed in this section. The only difference between the first two methods is the way the macro that processes the data is invoked. Method1 uses the SAS *call symput* routine to generate multiple macro calls. Method2 uses a *put* statement to write out macro calls to an external file, then includes that external file back into the program using a *%include* statement to invoke the macro. With the open code method, a block of SAS code which processes the data is generated for each observation in the parameter file using the parameter values. The SAS code is written out to an external file and then included back into the program for execution.

Processing with Macros: Method1

Method1 proceeds as follows:

1. Read the parameter file. The number of observations (or the number of data files to be processed) in the parameter file is stored into the macro variable FILENUM using the *call symput* routine. A macro invocation statement is generated for each observation using the values read in from the parameter file and stored into a macro variable with the corresponding observation number (FILE1, FILE2, ..., FILE.&FILENUM).
2. Write a macro to process the data using the variables in the parameter file as macro parameters. This macro may include a data step and any procedures that need to be used.

3. Write another macro to invoke the macro from Step 2 by looping through the macro calls generated in Step 1, with the value of the macro variable FILENUM as the total number of iterations.

The SAS code to implement the above steps follows:

/* Step 1: Read parameter file, declare FILENUM, the total number of observations, and create a macro call for each observation */

```
data _null_;
  set pfile end=last;
  if last then call symput('filenum',left(_n_));
  call symput ("file" || left(_N_),
    '%execute(' || strand || ',' || level || ',' || form || ',' || trim(dnum)
    || ',' || pnum || ',' || trim(dstart) || ',' || pstart ')');
```

The last *call symput* statement has the same effect as using a series of *%let* statements such as:

```
%let file1=(r,6,51,10,7,7,18);
%let file2=(r,6,52,10,5,7,16);
```

/* Step 2: write a macro that processes the data files */

```
%macro execute(strand,level,form,dnum,pnum,dstart,pstart);
```

```
data raw;
  infile dd(nes&strand.&level.s&form);
  input @&dstart (d1-d&dnum) ($1.) @&pstart (p1-p&pnum) ($1.);
```

```
proc print;
  var d1-d&dnum p1-p&pnum;
```

```
%mend execute;
```

/* Step 3: Invoking the macro of step 2 in a loop */

```
%macro loop;  
  %do i=1 %to &filenum;  
    &&file&i  
  %end;  
%mend loop;  
%loop
```

For each value of *i*, *&&file&i* resolves to a macro call declared in Step 1 and invokes the macro *execute*. For *i* = 1, *&&file&i* first resolves to

&file1

and then to

```
%execute(r,6,51,10,7,7,18)
```

which invokes the macro *execute* to process data for Form 51 of sixth grade reading.

Processing with Macros: Method2

Method2 differs from Method1 in that, instead of using the *call symput* routine to create several macro variables, a *put* statement is used to write macro invocation statements to an external file. This file is then included, using the *%include* statement, to invoke the macro that processes the data. The SAS code is given as follows (note that Step 2 is identical to Step 2 of Method1).

/* Step 1: Read parameter file and write macro calls out to an external file */

```
data _null_;  
  set pfile;  
  file outdata(invoke);  
  put '%execute('strand', 'level', 'form', 'dnum', 'pnum', 'dstart', 'pstart')';
```

The *invoke* member of the *outdata* file has the following look:

```
%execute(r,6,51,10,7,7,18)
%execute(r,6,52,10,5,7,16)
.....
%execute(m,10,69,30,4,9,40)

/* Step 2: write a macro that processes the data files */

%macro execute(strand,level,form,dnum,pnum,dstart,pstart);

data raw;
    infile dd(nes&strand.&level.s&form);
    input @@&dstart (d1-d&dnum) ($1.) @@&pstart (p1-p&pnum) ($1.);

proc print;
    var d1-d&dnum p1-p&pnum;

%mend execute;

/* Step 3: Include the output file of step 1 to invoke the macro that
processes data */

%include outdata(invoked);
```

Processing with Open Code: Method3

Method3 proceeds as follows:

1. Read the parameter file and create two new data step variables (INNAME and OUTNAME) by concatenating the values of the variables read from the parameter file. INNAME specifies the names of the external data files that are going to be used as input. OUTNAME is used to name output data sets.
2. For each observation in the parameter file, write a block of SAS code which processes data into an external file, based on the parameter values read in for each case.
3. Execute the code generated in Step 2 using *%include* statement at the end of the program.

The SAS code for implementing the above steps is given as follows:

/* Step 1: Read in parameter file and create two new variables */

```
data _null_;  
    set pfile;  
    outname=compress(strand||level||form);  
    inname=compress('nes'||strand||level||'s'||form);
```

/* Step 2: Writing the code that processes the data into an external file */

```
file data2(putfile);  
put 'data 'outname ' ;'  
' infile dd' '(' inname ')' ;' /  
' input @' dstart '(d1-d' dnum ' ) ($1.)'  
' @' pstart '(p1-p' pnum ' ) ($1.) ;'  
proc print ;  
' var d1-d' dnum ' p1-p' pnum ;' ;
```

The *put* statement writes the following code into an external file for the first observation from the parameter file:

```
data r651;  
    infile dd(nesr6s51);  
    input @7 (d1-d10) ($1.) @18 (p1-p7) ($1.) ;  
  
proc print;  
    var d1-d10 p1-p7;
```

/* Step 3: Execute the code generated in step 2 */

```
%include data2(putfile);
```

Performance Assessment

Each of the three methods described in the preceding section was used to process 50 data sets using Release 6.07 of SAS. The number of variables in each data set ranged from 14 to 52, and the number of observations in each file ranged from approximately 600 to 1100.

The performance statistics (CPU, I/O, and memory) were obtained for each method and are presented in Figure 1. The differences between Method1 and Method2 were minimal. Method3 was the best in terms of both CPU time and memory use. However, in terms of actual programming time, Method3 required much more programmer time than the other two methods. If computer resources (such as CPU time) are a big concern, Method3 might be preferred over the other two methods. Otherwise, Method1 or Method2 will produce very similar results with much less programming effort.

The simultaneous approach discussed in this section has at least three advantages. First, it facilitates file maintenance since the number of program files to be maintained is small and the programs can be easily adapted to different applications. For example, the macro *execute* used with Method1 and Method2 can include any SAS procedures to process the data if need be. Second, for data analysis involving a large number of data files, use of the simultaneous method can save a great deal of programming time by not having to write and copy multiple program files. Finally, the simultaneous method frees the programmer or data analyst from the boredom and tediousness of having to repeat the same process again and again.

SIMULTANEOUS PROCESSING WITH OTHER PROGRAMS

It is often the case that the same data need to be analyzed using different programs, many of which do not have the flexibility and versatility of the SAS system. In the current application, the data need to be processed using a mainframe package MSTEPS, a program used for test calibration using Rasch item response theory (Wright, Congdon, & Rossner, 1987). For each data set, there needs to be a separate file containing the necessary JCL statements, Msteps control statements, and a separate data file to be accessed by MSTEPS. In this case, writing or copying JCL and MSTEPS control statement files (changing certain values each time) multiple times and preparing MSTEPS input files one by one can be extremely tedious and error prone. The same techniques that were discussed in the preceding section can be employed to write JCL and MSTEPS command statement files as well as the data files. The JCL and the MSTEPS files can be generated within one job and then the output JCL and MSTEPS files can be submitted in one job. This procedure includes several steps illustrated in Figure 2.

The code to implement the procedure are given in Appendix 1. Excerpts of SASTMP and JCLMSTEP are given in Appendix 2 and 3, respectively.

SUMMARY

Techniques for simultaneous processing of multiple files with different record layouts and different number of variables were explored in this paper. Essential to this approach is the creation of a parameter file which contains all the necessary information with regard to the differences in the layout and which provides parameter values for generating flex code. Two methods of using macros and one method of using open code were described and compared for efficiency. While the open code method was found to be the more efficient method in terms of computer resources, it took more programming time. Also discussed was the way in which SAS can be used to generate JCL and program control language to make simultaneous processing possible with other mainframe programs as well.

The advantages of the simultaneous method include (a) facilitating file maintenance, (b) considerable savings in programming time, and (c) relief from the boredom and tediousness of having to mechanically repeat the same process.

REFERENCES

- Aster, R., & Seidman, R. (1991). *Professional SAS Programming Secrets*. New York: McGraw-Hill, Inc.
- Wright, B.D., Congdon, R. T., & Rossner, M. (1987). *MSTEPS: A Rasch Program for Ordered Response Categories*. Chicago: MESA Press.

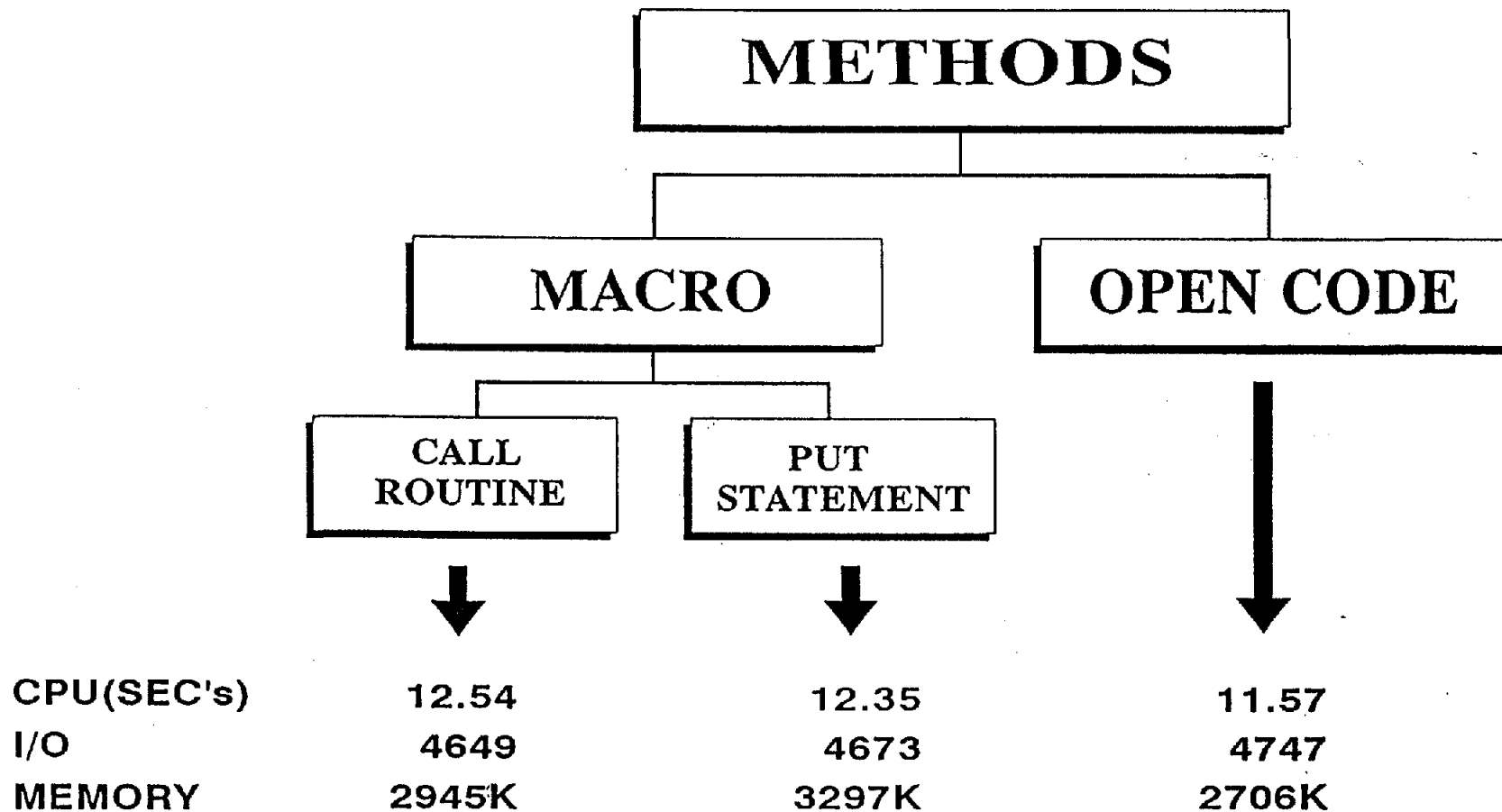


FIGURE 1: Methods for Simultaneous Processing with SAS and Performance Assessment

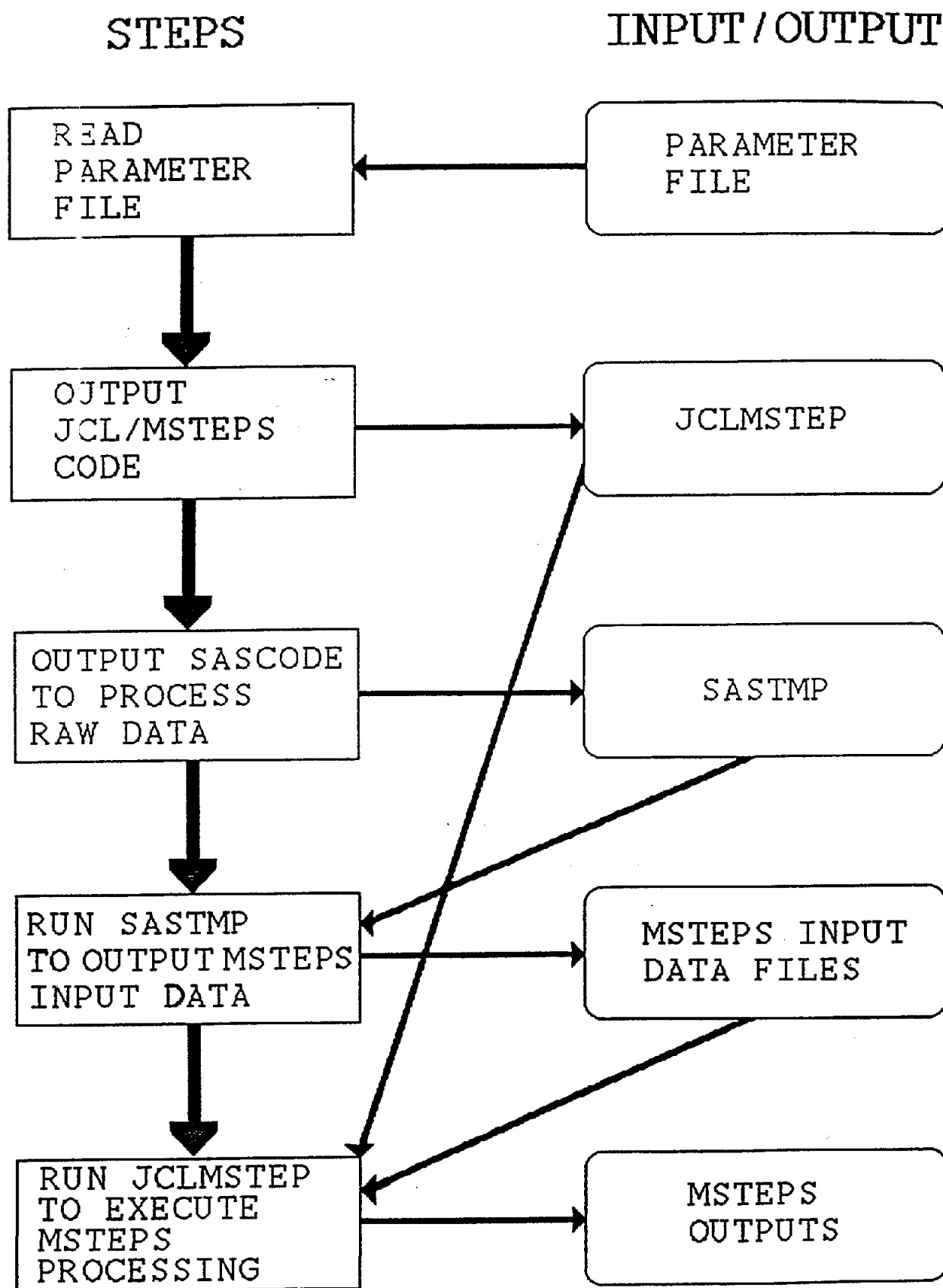


FIGURE 2. SIMULTANEOUS PROCESSING
WITH MSTEPS

APPENDIX 1. Code for simultaneously processing multiple files with MSTEPS

**/*step1: read the parameter file and create
input/output data set names */**

```
data _null_;
  set pfile;
  outname=compress(STRAND||LEVEL||FORM);
  inname=compress('NES'||STRAND||LEVEL||'S'||FORM);
  tmpname=outname||',DISP=OLD';
  inum=dnum+pnum;
  space=pstart-dnum-1;
```

**/*step2: Write JCL/MSTEPS code for each test form in
an output file called 'JCLMSTEP'*/**

```
file out(JCLMSTEP);
```

/*write jobname statement at the beginning of JCLMSTEP*/

```
if _n_=1 then put
  '//PTAHT1C1 JOB(MPTA6014),'HUIXING TANG',NOTIFY=PTAHT1,"/
  '//      MSGCLASS=T,MSGLEVEL=(1,1),CLASS=A";
```

/*write JCL/MSTEPS codes for each case (test form)*/

```
put
  '//inname 'EXEC PGM=MSTEPS,REGION=2048K          '/
  '//STEPLIB DD DSN=PMGCE1.LINKLIB,DISP=SHR'/'
  '//      DD DSN=SYS1.VFORTLIB,DISP=SHR'/'
  '//FT05F001 DD DDNAME=SYSIN'/'
  '//FT06F001 DD DSN=PTAHT1.NES.EXEC(S' tmpname /
  '//FT08F001 DD DSN=&&TEMP1,DISP=(NEW,PASS),UNIT=SYSDA,'/
  '// SPACE=(CYL,(10,2)),DCB=(RECFM=FB,LRECL=120,BLKSIZE=12000)'/
  '//PRMFILE DD *                               /
  ' &INST  '/
  ' NI=' inum /
  ' ITEM1=8'/'
  ' INAMES=1'/'
  " FORMAT='(A80)'  "/
  " CODES='012'" /
```


**/*step 4: Execute the SAS code written in step 3 to process
each raw data set and write out the MSTEPS input file */**

%include out(SASTMP);

**/*step 5: Exit to TSO and Submit the JCLMSTEP file
written in step 2. */**

submit out(JCLMSTEP)

APPENDIX 2. Excerpts of SASTMP

```
DATA R651 ;
  INFILE DD (NESR6S51 );
  INPUT @7 (Q1-Q17 ) (10 *$1. +1 7 *$1.);
  ID= N_ ;
    ARRAY Q(17 ) $1. Q1-Q17 ;
    ARRAY I(17 ) 2. I1-I17 ;
  DO A=1 TO 17 ;
    IF A <= 10 THEN DO;
      IF Q(A)='A' OR Q(A)='B' OR Q(A)='C' OR Q(A)='D' THEN I(A)=1;
    ELSE IF Q(A)='1' OR Q(A)='2' OR Q(A)='3' OR Q(A)='4' THEN I(A)=0;
    END;
  ELSE DO;
    IF Q(A)='1' THEN I(A)=0;
    IF Q(A)='2' THEN I(A)=1;
    IF Q(A)='3' THEN I(A)=2;
    IF Q(A)='4' THEN I(A)=3;
  END;
END;
```

```
DATA NULL_ ;
  SET R651 ;
  FILE DATA2(DR651 );
  PUT (ID I1-I17 )(Z7. 17 *1.);
```

```
DATA R652 ;
  INFILE DD (NESR6S52 );
  INPUT @7 (Q1-Q15 ) (10 *$1. +1 5 *$1.);
  ID= N_ ;
    ARRAY Q(15 ) $1. Q1-Q15 ;
    ARRAY I(15 ) 2. I1-I15 ;
  DO A=1 TO 15 ;
    IF A <= 10 THEN DO;
      IF Q(A)='A' OR Q(A)='B' OR Q(A)='C' OR Q(A)='D' THEN I(A)=1;
    ELSE IF Q(A)='1' OR Q(A)='2' OR Q(A)='3' OR Q(A)='4' THEN I(A)=0;
    END;
  ELSE DO;
    IF Q(A)='1' THEN I(A)=0;
    IF Q(A)='2' THEN I(A)=1;
```

IF Q(A)='3' THEN I(A)=2;
IF Q(A)='4' THEN I(A)=3;
END;
END;

....
DATA _NULL_;
SET R652 ;
FILE DATA2(DR652);
*PUT (ID 11-115)(Z7. 15 *1.);*

[illegible]

