

Teacher Evaluations - A tutorial in some advanced features of SAS

Glenn Millard
Stephen F. Austin State University

A number of years ago Stephen F. Austin State University started having the students evaluate their teachers at the end of the semester. At the time the entire University used a common questionnaire on an Opscan sheet that was scanned and scored with a simple SAS program by our IT Operations staff. A few years ago the common questionnaire was abandoned and each Department or School started using their own questionnaires. To score these various evaluations we ended up with a number of SAS programs, each slightly different, to handle each version. In an effort to simplify the scoring process for our Operations staff, a single more complex SAS program has been developed to handle all of the various versions. Now, once various parameter files have been updated for the semester, the process of scoring all of the various versions of our teacher evaluations is a rather simple process.

The evaluation program started as a simple one-step SAS program that read just the one datafile. The new version has two major programming steps, uses 5 auxiliary files, uses SAS macros, a DDE link to Excel to print the output, and SAS Connect to update a master file on a minicomputer running VMS where all the evaluation data will be stored so reprints or further statistics can be run later. This rather simple application is now a good example for novice SAS programmers on how to use a number of the more advanced features of SAS.

Introduction

Stephen F. Austin State University's teacher evaluation program started a number of years ago when I helped design a custom evaluation form for our NSC Opscan scanner and developed a SAS program to summarize and print the results after the sheets had been scanned. The SAS program was designed to handle multiple classes per datafile. Each class is separated by a specially marked Opscan sheet. A number of years later the University decided to drop the standardized evaluation form and let each department design their own evaluations. The SAS program was modified to allow a variable number of questions, I set the maximum to 60. Due to the large arrays in the SAS program, only about 10 classes could be scanned in one group. When I got a request to summarize all the classes in one department in one group I quickly found a problem. I was creating a huge array to store the data since I had no idea when the class was scanned how many students were in the class. I was allowing for a maximum of 60 questions and 300 students per class. When I got the request to summarize the entire department, with over 1000 students, I quickly found out that SAS could not create an array that size because it created more variables, over 32,000, than the SAS data vector in SAS 6.12 could handle. I was calculating the mean, standard deviation, and standard error by called SAS functions in the data step. To get around the array problem, I now needed to calculate these values with the SAS Means Procedures and then merge this data with Proc Freq output in the rest of the program. I needed a complete rewrite of my program. Because I was using Put statements to output the results, even Proc Tabulate didn't give me the exact output I wanted, my output looked like something out of the 60's. By this time I now had SAS 8.2 available and the possibility of using ODS to enhance the output. After finding out that even ODS wouldn't give me exactly the look I wanted I looked for other solutions. After solving a few interesting problems, and learning a lot more about some SAS features I have the program we are using now. Parts of the code in the program serve as good examples of SAS features that can be quite useful to those trying to learn some of the more advanced programming features of SAS.

Multiple Missing Values

One very helpful feature of SAS is its ability to allow you to assign multiple missing values. Why would you want to do this? This program shows an example of why.

In scoring the evaluations via an Opscan sheet with 5 responses (A through E) I normally code the responses 1 through 5. The Opscan returns a zero for a unmarked response. When I got a request from a department to score the responses 4 through 0 I now had a problem. How was I to distinguish non-response, with a value of zero, from a valid zero score? In this case the values would get reported in different columns in the report. I was able to do this by coding multiple missing values. To assign a normal missing value to a variable you can use this code:

```
ques=.
```

The period is used to assign a missing value to ques. You can use the following code to assign a special missing value to a variable.

```
ques=.A
```

Any letter or an underscore can be used. In this case I set a non-response to .A and a valid zero response as .B. Later in the program I can then distinguish these two values and post the proper counts and percentages to the right columns.

Secondary File Access

The old version of the teacher evaluation program got all of the class identification information from the Opscan sheets. This information consisted of department, course, section, and teacher number. From the beginning we have had problems with getting this coded correctly on the sheets. In an effort with the new program to be more accurate we are now coding a 5-digit call number on the sheets that uniquely identifies the class. This is the same code used by the students to register for the class. Now a secondary file is used to cross reference this number with the proper department, course, section, and teacher name. This file is updated each semester for the current courses. This file has been setup as an indexed file and is accessed with the following code:

```
set gsaseval.allspring1 key=callnum;  
if _error_ then do;  
    put 'missing callnum' callnum;  
    stop;  
end;
```

The automatic SAS variable `_error_` is used to determine if the matching call number is found in the file. `_Error_` will be set to 1 if the record is not found. If the call number is not found the call number is printed to the log and the program is stopped.

Once the course has been properly identified another secondary file is accessed with departmental information to determine specific scoring information for the class. Information with scoring values and labels, and the number of questions to be scored and their labels are in this file. There is one record for department in this file. Some specific problems occurred when trying to access this file as an indexed file.

In most cases multiple classes for a single department will be run together as one group. This file needs to be accessed once each time a new class starts processing. If the new class was from the same department as the last class an error would result. SAS would get a record not

found error even though the record was found for the last class. Displaying the automatic variable `_IORC_` revealed this to the case. This would not be a problem except for the fact that even with a Retain statement to retain the values from their read values would not work either. I did not check with SAS Support on this issue, but I believe this occurs because the record is still locked and because it is not the primary file the Retain statement will not keep the values. While this could be a serious problem in some languages, not so in SAS. There is another way to solve this problem in SAS and that is to use the `POINT=` option on the Set statement to read the dataset by observation number. In this case, since the file is small, a simple loop is set to scan from the front of the file until the record is found. If it does not find a match it will use the parameters for 'UNK', which is the last record in the file.

```
found=0;
do until (found=2);
    j+1;
    set gsaseval.evalconf point=j;
    if dept='UNK' then found=2;
    if dept=deold then found=2;
end;
```

LINK Statement

Another data step statement used in the program is the Link statement. I have not seen this statement used often, but it is very easy to use. It works like a Perform statement in COBOL. You use a Label statement and a Return statement with the Link statement. SAS will branch to the specified Label on the Link statement as execute all SAS statements until a Return statement is found. At that time SAS will return and execute the next statement below the Link statement. Link statements can be nested. You should use a Return statement just before the Label statement to make sure SAS does not execute these statements by accident. Example code would be:

```
data _null_;
```

SAS statements ...

```
Link printit;
```

SAS statements ...

```
Return;
printit;;
```

SAS statements ...

```
Return;
run;
```

Somewhere in your data step you will probably need a Stop statement to stop the Data Step execution at that point.

SAS CONNECT

The SAS Connect product allows SAS on a local host to work in a client/server environment with a server running SAS. In this case the local host is a PC running SAS on Windows XP and a minicomputer from Compaq running SAS under VMS. The first step in using SAS Connect is to start the connection. SAS provides connect scripts with SAS Connect to connect to any server supported by SAS. In this case the script is TCPVMS.SCR. The following statements start the connection to our host, named TITAN.

```
Options remote=TITAN;
filename rlink '!sasroot\connect\saslink\tcpvms.scr';
signon;
```

In this program we are updating a master file containing all previously scanned evaluations with the additional evaluations in the current group. Both data sets must be sorted before the update can be done. First the local data set is sorted.

```
Proc sort data=stueval nodupkey;
by term callnum studseq;
run;
```

Then using Compute Services with RSUBMIT in the SAS Client/Server environment the remote master file is sorted. Note that the EVAL Libname used in the remotely submitted sort is a SAS Libname that has been previously established by the VMS host.

```
Rsubmit;
proc sort data=eval.stueval nodupkey;
by term callnum studseq;
run;
endrsubmit;
```

Now that the data sets are sorted a local Libname needed to be established to the same location as the remote Libname EVAL. Therefore we locally submit the following statement:

```
libname sast remote 'titan$dra4:[supportgrp.evaluations.eval]' server=titan;
```

In this case at least the link to the remote directory should be a VMS physical address. VMS allows logical addresses but they do not work in all cases. It is now time to update the remote data set. The following data step is submitted locally to SAS:

```
data sast.stueval;
  modify sast.stueval work.stueval;
  by term callnum studseq;
  if _iorc_=%sysrc(_sok) then replace;
  else if _iorc_=%sysrc(_dsenmr) then
    do;
      output;
      _error_=0;
    end;
  else
    do;
      put "Unexpected error at Observation: " _n_;
      _error_=0;
      stop;
    end;
```

```
run;
```

In this example the automatic SAS variable `_iorc_` is tested to see if a duplicate record exists on the master file. A status set to `_sok` means the record does exist and is then replaced. A status set to `_dsenmr` means the record doesn't exist and the new record is output to the data set.

Output Data Sets

As mentioned in the introduction, Proc Means and Freq are used to calculate means, standard deviation, standard error, counts and percentages for each question. To do this output data sets were created from the procedures rather than printed output. Note that Proc Means can only create a data set with one statistic for each variable. Proc Freq can create a data set with multiple statistics, but for only one variable.

```
proc means data=stueval noprint;
by testseq callnum notsorted;
output out=stumean mean=;
run;

proc freq data=stueval;
by testseq callnum teach notsorted;
tables ques1 / missing noprint out=stfreq1;
```

SAS Macros

Since I have allowed for up to 40 questions on a teacher evaluation, this means 40 Proc Freq would need to be run 40 times and then the 40 data sets merged. A simple way to code this is to use a SAS macro. The macro code to do this looks like this:

```
%macro freqtab;
proc freq data=stueval;
    by testseq callnum teach notsorted;
    %do n=1 %to 40;
        tables ques&n / missing noprint out=stfreq&n;
    %end;
run;
proc datasets library=work memtype=data;
    delete totfreq;
run;
%do i=1 %to 40;
data freqmerg;
    set stfreq&i;
    quest=&i;
    val=ques&i;
    dcallnum=callnum;
    cnt=count;
    drop ques&i count callnum;
run;
proc append base=totfreq data=freqmerg;
run;
%end;
%mend;
```

In this code the macro variable `n` is used to create 40 data sets. We then make sure total data set doesn't exist before the following two steps are run 40 times to clean up the data and

then merge them together with Proc Append.

The other use of the use of a macro in the program is to trap for errors in the first data step procedure and stop the execution of the following procedures. In this case we are only worried about the 5-digit class call number. If a call number is missing or incorrect we want to stop execution of the following procedures and then display where the error has occurred. Our evaluation program consists of a data step that reads the data, multiple procedures, and finally a data step that prints the output. Since a macro must be defined before it is executed we need to enclose all but the first data step in the macro statements and move them ahead of the first data step. If we find a missing or incorrect call number we need to pass that value and a flag to stop further execution to the macro. We do this by using the CALL SYMPUT statement to pass these values to a SAS macro variable from the data step. At the start of the macro we use an %IF statement to test for an error. If there is not an error we execute the procedures. If there is an error we use an %ELSE statement and use a %PUT _user_ statement to output all local created macro variables. With these statements at the bottom of the macro it makes seeing the error and its location easy since the values are displayed at the bottom of the log.

The general form of the macro is as follows:

```
%macro results;
%if &kount = 0 %then
  %do;
```

Proc and Data steps ...

```
%end;

%else
  %do;
  %put Miscoded Call number (KALLNUM) and scan group (TSEQ);
  %put _user_;
  %end;

%end;

%mend;
```

```
data stueval;
```

Data step statements ...

```
call symput('kount',0);
if studseq=1 then do;
  code=right(id);
  callnum=code;
  teach=birthisr;
  set gsaseval.allspring1 key=callnum;
  if _error_ then do;
    put 'missing callnum ' callnum;
    call symput('kount',1);
    call symput('kallnum',callnum);
    call symput('tseq',testseq);
  stop;
end;
```

Data step statements ...

```
run;  
  
%results;
```

SAS and DDE

For years I had wanted to make the output from the teacher evaluations look better. The output looked like something printed on a line printer from back in the 60's. Because none of the SAS procedures would print the output exactly the way I wanted, I had been using PUT statements to output the results. With SAS being able to use DDE to link to programs like EXCEL and then input or output data to that program I found a solution for my problem. The first thing I did was to design a custom blank EXCEL spreadsheet into which I would put the SAS output for the faculty evaluations. The EXCEL spreadsheet looked like this:

Callnum

Course

Instructor

Term

Number of Students

0

Response Key:

Question	Response Missing	Response (1)	Response (2)	Response (3)	Response (4)	Response (5)	Response (6)	Mean	Standard Deviation	Standard Error
1										
Percent	0	0	0	0	0	0	0	0.00	0.00	0.00
		0.00	0.00	0.00	0.00	0.00	0.00			
2										
Percent	0	0	0	0	0	0	0	0.00	0.00	0.00
		0.00	0.00	0.00	0.00	0.00	0.00			
3										
Percent	0	0	0	0	0	0	0	0.00	0.00	0.00
		0.00	0.00	0.00	0.00	0.00	0.00			
4										
Percent	0	0	0	0	0	0	0	0.00	0.00	0.00
		0.00	0.00	0.00	0.00	0.00	0.00			
5										
Percent	0	0	0	0	0	0	0	0.00	0.00	0.00
		0.00	0.00	0.00	0.00	0.00	0.00			
6										
Percent	0	0	0	0	0	0	0	0.00	0.00	0.00
		0.00	0.00	0.00	0.00	0.00	0.00			
7										
Percent	0	0	0	0	0	0	0	0.00	0.00	0.00
		0.00	0.00	0.00	0.00	0.00	0.00			

The spreadsheet spans multiple pages and allows for all 40 questions. A secondary SAS file has been setup to control where the row the data is put into and also to control the value of the last row to be printed. That way the page breaks are handled successfully and only the proper number of questions get printed.

Before the output data step is executed Excel needs to be started. The SAS X command does this. Note that it must point to Excel.exe, wherever that is located on your PC. After that SAS program execution is paused to allow Excel to start. Also at some point in the program the following SAS options should be set and a macro variable assigned to represent a Tab character. The Tab character is most important since it must be sent to Excel after data is written to each cell to release the cell. Otherwise your program will stall at this point.

```
OPTIONS NOXWAIT NOXSYNC;
%let tab='09'x;
```

```
X "c:\excel";
```

```
DATA _NULL_;
    rc = SLEEP(10);
RUN;

FILENAME ddedata DDE "excel|system" notab;
```

After the pause to allow Excel to start the Filename statement is used to assign the DDE link to Excel. The following File statement in the data step redirects the output of the following Put statements to Excel rather than the SAS Log.

```
FILE ddedata;
```

The following Put statements make sure no other spreadsheets are open and then opens the pre-formatted spreadsheet.

```
put '[error(false)]';
put '[Close.All()]';
put '[App.Restore()]';
put '[App.Maximize()]';
put '[open("C:\Shared Files\evaluations.xls")]';
```

To write data to Excel takes two Put statements. The first Put statement sends a Select statement to Excel to tell Excel what cell you will send output to. The next Put statement outputs the data and then sends a tab to release the cell. In all the examples I have seen the cell is referenced by row number and column number.

```
put '[select("R2C1")]';
put dcallnum &tab;
```

When I am writing out a row of data for a question into a row of cells you need to just select the position of the first cell. As I mentioned above I have a file that controls the proper row that I need to write to. In this case rather than having an absolute location to write to I need to calculate the position and create a variable to send to Excel that contains the proper position. You can string the command together that you will send to Excel. The one thing that must be done is after you string together the command you need to use the compress function to eliminate blanks in the string. Excel will not accept any blanks in the command string. The following is an example of writing out this data:

```
putstr = '[select("R' || printrow || 'C2:R' || printrow || 'C11")]';
putstr=compress(putstr);
put putstr;
put knt(1) &tab knt(2) &tab knt(3) &tab knt(4) &tab knt(5) &tab
    knt(6) &tab knt(7) &tab xmean(i) &tab stand(i) &tab stder(i) &tab;
```

After all the data is written to the proper cells it is time to either save or print the spreadsheet. This is where I ran into an interesting problem that I had to call SAS Support on. As Excel users are aware you can refer to a range of cells in Excel to two ways. In all our previous examples we have shown the row/column specification like R5C5 to refer to Row 5 Column 5. You can also use E5 to refer to this same cell. In all the above examples all I needed to do was refer to a simple starting cell position. Now I needed to specify a cell range to be selected before I printed or saved the file. I quickly found out that a range specification like R1C1..R20C6 would not work. To specify this range properly you must instead send A1..F20. Since my print range varies I again strung together the command to send to Excel.

```
putstr = '[set.print.area("a1..k' || printrow || '")]';
putstr=compress(putstr);
```

```
put putstr;  
put '[print()]';
```

The following command is an example of how to save the file.

```
PUT '[save.as("c:\Shared Files\dde3.xls")]';
```

The Close All command and the Open command, to open a fresh spreadsheet, shown earlier should be executed as part of each loop to make sure you start with a clean spreadsheet for each one you plan to create. After you finish things up all you need to do is branch to a statement to send a Quit command to Excel to tell it to shutdown.

```
PUT '[quit()]';
```

Because you issued the App.Maximize command to Excel you get watch SAS open and close the spreadsheet and load the data into the cells. It's interesting to watch. Your PC will be completely tied up while this is happening so sit back and enjoy your handiwork.

Conclusion

The flexibility of the SAS system and the SAS language helps make this application work. While some of the points shown above can take some time to master and feel comfortable using they can be quite useful in creating your applications. There are a couple of other SAS programs not shown in this paper that create some of the external files. They were not shown here because they are easy to write and can vary greatly depending on the data you are working with. This program was a good learning experience for me and should be for others.

The general program design can be quite useful in a number of similar applications. If you have an application where you are inputting data, checking it against various other existing files, creating statistics for this additional data and printing a concise printout of it or filling out a custom form to be printed, and finally updating a master data set; then take a closer look at this program.

References

SAS Language Reference: Dictionary, Version 8

SAS Procedures Guide, Version 8

SAS Macro Language: Reference, Version 8

SAS/CONNECT User's Guide, Version 8

"Using DDE to Communicate between SAS and Excel", James Hoffman, Proceedings of The 13th Annual South-Central SAS User's Group Regional Conference

Appendix 1 - Sample Output

Callnum 10375	Course BIO 238L021			Instructor Garrett M		Term 20042		Number of Students 17		
	Response Key:									
	1=Excellent			2=Very Good			3=Good			
	4=Fair			5=Poor			6=not used			
Question	Response Missing	Response (1)	Response (2)	Response (3)	Response (4)	Response (5)	Response (6)	Mean	Standard Deviation	Standard Error
1	What is your current standing in this course?									
Percent	0	1 5.88	4 23.53	6 35.29	5 29.41	1 5.88	0 0.00	3.06	1.03	0.25
2	How often did you miss class?									
Percent	0	11 64.71	5 29.41	1 5.88	0 0.00	0 0.00	0 0.00	1.41	0.62	0.15
3	How much time did you spend on this course outside of class time average?									
Percent	0	0 0.00	8 47.06	9 52.94	0 0.00	0 0.00	0 0.00	2.53	0.51	0.12
4	Was this course an enriching educational experience?									
Percent	0	13 76.47	3 17.65	1 5.88	0 0.00	0 0.00	0 0.00	1.29	0.59	0.14
5	How many times did you make appointments with your instructor for help or advice?									
Percent	0	0 0.00	0 0.00	5 29.41	12 70.59	0 0.00	0 0.00	3.71	0.47	0.11
6	Were the course requirements and testing and grading policies handed out and explained by the instructor?									
Percent	0	17 100.00	0 0.00	0 0.00	0 0.00	0 0.00	0 0.00	1.00	0.00	0.00
7	Was this course required in your major or minor?									
Percent	0	0 0.00	16 94.12	1 5.88	0 0.00	0 0.00	0 0.00	2.06	0.24	0.06
8	How many college credits have you completed?									
Percent	0	11 64.71	4 23.53	1 5.88	1 5.88	0 0.00	0 0.00	1.53	0.87	0.21

Appendix 2 - SAS Teacher Evaluation Program

```
%macro results;
%if &kount = 0 %then
  %do;
proc means data=stueval noprint;
by testseq callnum teacher notsorted;
output out=stumean mean=;
run;
proc means data=stueval noprint;
by testseq callnum teacher notsorted;
output out=stustd std=;
run;
proc means data=stueval noprint;
by testseq callnum teacher notsorted;
output out=stuerr stderr=;
run;
  %macro freqtab;
  /*data _null_;
set stumean;
call symput('kount',count);
call symput('testnum',testseq);
stop;
run;
  */
proc freq data=stueval;
by testseq callnum teacher notsorted;
%do n=1 %to 40;
tables ques&n / missing noprint out=stfreq&n;
%end;
run;
proc datasets library=work memtype=data;
delete totfreq;
run;
%do i=1 %to 40;
data freqmerg;
set stfreq&i;
quest=&i;
val=ques&i;
dcallnum=callnum;
cnt=count;
drop ques&i count callnum;
run;
proc append base=totfreq data=freqmerg;
run;
%end;
%mend;
%freqtab
quit;
proc sort data=totfreq;
by testseq quest val;
run;
/* proc datasets library=work;
  modify totfreq;
  index create grpq=(dcode quest);
run;
*/
/* Invoke Microsoft Excel. */
```

```

X "c:\excel";

DATA _NULL_;
    rc = SLEEP(10);
RUN;

FILENAME ddedata DDE "excel|system" notab;

title1 ' ';
options pagesize=55 linesize=132 nodate nonumber;
/* filename foo printer recfm=p; */
data _null_;
array pct(7) pct1-pct7;
array knt(7) knt1-knt7;
array ftot(240,7) ftot1-ftot1680;
array fpct(240,7) fpct1-fpct1680;
array xmean(240) xmean1-xmean240;
array stand(240) std1-std240;
array stder(240) stder1-stder240;
array ques(240) ques1-ques240;
array q(40) q1-q40;
array qv (6) qv1-qv6;

attrib dept deold length=$3. callnum length=5.0 sem length=5.0;
retain deptcode j a std1-std240 stder1-stder240 xmean1-xmean240 sem
        ftot1-ftot1680 fpct1-fpct1680 k deold qv1-qv6 callnum code;
attrib putstr length=$60. quesnum length=4.0;

set totfreq end=last;
by testseq;

if last then do;
    link printit;
    link closeit;
end;

if first.testseq then do;
    do m=1 to 240;
        do n=1 to 7;
            ftot(m,n)=0;
            fpct(m,n)=0;
        end;
    end;
    do m=1 to 240;
        xmean(m)=0;
        stand(m)=0;
        stder(m)=0;
    end;

    set stumean;

    a=count;

    k=_freq_;
    callnum=dcallnum;
    set gsaseval.allspring1 key=callnum;
    if _error_ then do;
        put 'missing callnum ' callnum;
        stop;
    end;
end;

```

```

end;
  deold=dept;
  sem=term;

  do i=1 to a;
    xmean(i)=ques(i);
  end;

  set stustd;
  do i=1 to a;
    stand(i)=ques(i);
  end;
  set stuerr;
  do i=1 to a;
    stder(i)=ques(i);
  end;

  deold=dept;
  j=0;
  found=0;
  do until (found=2);
    j+1;
    set gsaseval.evalconf point=j;
    if dept='UNK' then found=2;
    if dept=deold then found=2;
  end;

end;

do b=1 to 6;
  if val = qv(b) then c=b;
end;

if val=.A then c=0;
if val=.B then c=0;

if val=. then c=0;

c=c+1;
ftot(quest,c)=ftot(quest,c)+cnt;
fpct(quest,c)=fpct(quest,c)+percent;

if last.testseq then do;

  link printit;
end;

return;

printit;;

  deold=dept;
  j=0;
  found=0;
  do until (found=2);
    j+1;
    set gsaseval.evalconf point=j;
    if dept='UNK' then found=2;
    if dept=deold then found=2;
  end;

```

```

FILE ddedata;

put '[error(false)]';
put '[Close.All()]';
put '[App.Restore()]';
put '[App.Maximize()]';
put '[open("C:\Shared Files\evaluations.xls")]';
put '[select("R2C1")]';
put dcallnum &tab;
put '[select("R2C3")]';
put crssect &tab;
put '[select("R2C5")]';
put instructor &tab;
put '[select("R2C7")]';
put sem &tab;
if teacher>1 then do;
  put '[select("R1C9")]';
  put 'Teacher' &tab;
  put '[select("R1C10")]';
  put teacher &tab;
end;
put '[select("R3C10")]';
put k &tab;

put '[select("R6C2")]';
put qt1 &tab;
put '[select("R6C5")]';
put qt2 &tab;
put '[select("R6C8")]';
put qt3 &tab;
put '[select("R7C2")]';
put qt4 &tab;
put '[select("R7C5")]';
put qt5 &tab;
put '[select("R7C8")]';
put qt6 &tab;

if nques = 0 then do;
  k=0;
  found=0;
  do until (found=2);
    k+1;
    set gsaseval.allcconf point=k;
    if _error_ then do;
      put 'missing callnum ' callnum;
      stop;
    end;
    if dcallnum=callnum then found=2;
  end;
end;

end;

pt=0;
do i=1 to a;
  pt+1;
  set gsaseval.excelnum point=pt;

  do ii=1 to 7;
    knt(ii)=ftot(i,ii);

```

```

    pct(ii)=fpct(i,ii);
end;

    quesnum=q(i);
    if quesnum > 0 then do;
        set gsaseval.question key=quesnum;
        prow=printrow-1;
        putstr='[select("R" || prow || "C2")]';
        putstr=compress(putstr);
        put putstr;
        put question &tab;
    end;

    putstr='[select("R" || printrow || "C2:R" || printrow || "C11")]';
    putstr=compress(putstr);
    put putstr;
    put knt(1) &tab knt(2) &tab knt(3) &tab knt(4) &tab knt(5) &tab
        knt(6) &tab knt(7) &tab xmean(i) &tab stand(i) &tab stder(i) &tab;

    printrow+1;
    putstr='[select("R" || printrow || "C3:R" || printrow || "C8")]';
    putstr=compress(putstr);
    put putstr;
    put pct(2) &tab pct(3) &tab pct(4) &tab pct(5) &tab pct(6)
        &tab pct(7) &tab;

end;

    putstr='[set.print.area("a1..k" || printrow || "")]';
    putstr=compress(putstr);
    put putstr;
    put '[print()]' ;
/*      PUT '[save.as("c:\Shared Files\dde3.xls")]';    */
return;

closeit;;

    PUT '[quit()]';
    stop;

run;
/* Login to Titan */
options remote=titan;
filename rlink '!sasroot\connect\saslink\tcpvms.scr';
signon;

/*
Assign Titan libname for datafiles
*/

libname sast remote 'titan$dra4:[supportgrp.evaluations.eval]' server=titan;

proc sort data=stueval nodupkey;
by term callnum studseq;
run;

rsubmit;

proc sort data=eval.stueval nodupkey;

```

```

by term callnum studseq;
run;

endrsubmit;

data sast.stueval;
  modify sast.stueval work.stueval;
  by term callnum studseq;
  if _iorc_=%sysrc(_sok) then replace;
  else if _iorc_=%sysrc(_dsenmr) then
    do;
      output;
      _error_=0;
    end;
  else
    do;
      put "Unexpected error at Observation: " _n_;
      _error_=0;
      stop;
    end;
  end;
run;

signoff;

quit;

%end;

%else
  %do;
    %put Miscoded Call number (KALLNUM) and scan group (TSEQ);
    %put _user_;
  %end;

%end;

%mend;

OPTIONS NOXWAIT NOXSYNC;
  %let tab='09'x;
  data stueval;

infile 'c:\Shared Files\staneval-test.sdf' pad missover end=last lrecl=424;
length code $9.;
array ans (240) ans1-ans240;
array ques (240) ques1-ques240;
attrib dept deold length=$3. count length=4.0 callnum length=5.0 teacher
length=3.0;

retain code count deold callnum teacher;
/* retain form exam; */

input @29 divider $ 6. @;

if divider = '999999' then do;
  studseq=0;
  testseq+1;
delete;

```

```

end;

studseq+1;

input @1 name $ 19. id $ 9. spec1 6. sex $ 2. birthyr $ 2. birthmo $ 2.
      (grade1-grade4) ($ 1.) resv1 $ 1. resv2 $ 1. (ans1-ans240) (2.);
/*
input @12 id $ 9. form $ 1. exam $ 3. (ans1-ans200) (2.);
*/

call symput('kount',0);
if studseq=1 then do;
  code=right(id);
  callnum=code;
  teacher=birthyr;
  set gsaseval.allspring1 key=callnum;
  if _error_ then do;
    put 'missing callnum ' callnum;
    call symput('kount',1);
    call symput('kallnum',callnum);
    call symput('tseq',testseq);
    stop;
  end;
end;

if crselect = 2 then do;
  k=0;
  found=0;
  do until (found=2);
    k+1;
    set gsaseval.allcconf point=k;
    if _error_ then do;
      put 'missing callnum ' callnum;
      call symput('kount',1);
      call symput('kallnum',callnum);
      call symput('tseq',testseq);
      stop;
    end;
    if code=callnum then found=2;
  end;
  if nques=0 then count=spec1;
  else count=nques;
end;

if crselect = 1 then do;
  deold=dept;
  k=0;
  found=0;
  do until (found=2);
    k+1;
    set gsaseval.evalconf point=k;
    if dept='UNK' then found=2;
    if dept=deold then found=2;
  end;
  if nques=0 then count=spec1;
  else count=nques;
end;

array qv (6) qv1-qv6;
/*
  form1=form;
  exam1=exam;
*/

```

```

/*
input @1 name $ 19. id $ 9. spec1 3. spec2 3. sex $ 2. birthyr $ 2. birthmo $
2.
      (grade1-grade4) ($ 1.) resv1 $ 1. resv2 $1. (ans1-ans240) (2.);

input @12 id $ 9. form $ 1. exam $ 3. (ans1-ans200) (2.);
*/
end;
end;
do i=1 to count;
  ques(i)=.A;
  if ans(i) = '3' then ques(i) = qv(6);
  if ans(i) = '16' then ques(i) = qv(5);
  if ans(i) = '8' then ques(i) = qv(4);
  if ans(i) = '4' then ques(i) = qv(3);
  if ans(i) = '2' then ques(i) = qv(2);
  if ans(i) = '1' then ques(i) = qv(1);
  if ans(i) = '0' then ques(i) = .B;
end;
do i=1 to count;
  qu=ques(i);
  if qu = 0 then ques(i)=0;
end;
keep callnum testseq studseq teacher dept crssect term ques1-ques40 count;
run;

%results;

```