

FIVE TECHNIQUES TO SIMPLIFY AND SPEED UP DATA _NULL_ REPORT PROGRAMMING

Tyler Cole, Pacific Data Designs, Inc., San Francisco, CA

ABSTRACT

SAS programmers are often asked to create numerous data reports by a challenging deadline. These reports may be based on sample reports which can only be recreated using DATA _NULL_. The task of completing a large number of reporting programs may seem laborious, even impossible given tight time constraints. In such situations, programming DATA _NULL_ reports in the shortest time possible becomes critical. This paper presents five techniques which will simplify DATA _NULL_ report writing thus reducing the time needed to program reports.

INTRODUCTION

Each of these techniques is individually described in its own section. The first three techniques address issues commonly encountered in report programming. These issues include:

- Creating an empty report when data is absent
- Aligning footers vertically across reports
- Numbering report pages with a total page count

The latter two techniques implement programming tools which facilitate report writing. These techniques include:

- Positioning headers and report entries using a heading ruler
- Formulating DATA _NULL_ statements using a macro interface.

TECHNIQUE I – CREATING AN EMPTY REPORT

What should a reporting program do if there is nothing to report? If no data is available for a DATA _NULL_ step, it may be necessary to create a report having no entries except a message stating an absence of data. When this is the case, consideration must be given to placement of the data-reading statement executed in a DATA _NULL_ step. Technique I – Creating An Empty Report describes the placement of this statement in addition to other statements needed to create an empty report when no observations are available.

A DATA _NULL_ step will complete one iteration for each observation input using data-reading statements such as SET, MERGE, UPDATE, and MODIFY. Consequently, no iterations will be completed if zero observations are input. In fact, the DATA _NULL_ step will terminate immediately after the data-reading statement has executed - no further statements are considered. The sample DATA _NULL_ step below takes this into account and begins by checking the number of observations available in the input dataset. If no observations exist, a header, explanatory message, and a footer are written to the report. Observations are then read from the input dataset. The DATA _NULL_ step will terminate if no observations are

available, otherwise the DATA _NULL_ step continues until the report is completed.

```
data _null_;
file MyFile print header=hdr;
if nobs=0 then do;
    put 'No Data Available';
    link ftr;
end;
set MyData nobs=nobs;

DATA _NULL_ statements...
return;

hdr:
header-creating statements...
return;
ftr:
footer-creating statements...
return;
run;
```

Please note that the nobs temporary variable holds the number of observations to be input. Its value is set at compile time and available when the DATA _NULL_ step begins execution.

TECHNIQUE II - ALIGNING FOOTERS

In general, a multi-page data report will be more visually appealing if report pages are similar in appearance. Likewise, a collection of data reports will be easier to read if individual reports share a common layout. Footers are one item affecting the similarity of reports in a collection. They may contain report-specific entries – such as how an entry was calculated – which are present for one particular report; footers may also contain report-standard entries - such as page numbering – which are found on all reports.

Report-specific footer entries can be aligned across report pages thereby improving readability. Report-standard entries can also be aligned across reports having the same effect. Aligning footers may seem trivial when dealing with a small number of reports having few pages; however, the improvement in appearance is noticeable when dealing with a large collection of multi-paged reports.

The sample code below shows one method of aligning footers across reports. A macro variable – ftrlines – holds the number of lines required by footer entries on the report. The value of this macro variable may vary across reports depending on the number of report-specific footer entries present. The DATA _NULL_ step keeps track of the number of page lines remaining as report entries are written. When this number equals that required for footer entries, a footer is inserted and a new page begun. A footer is also inserted, with appropriate positioning, when the last report observation is written to the report. This way, footers will be aligned across all pages on a report. Additionally, if report-specific footers are positioned above report-standard

footers, then report-standard footers will be aligned across all reports in a collection.

```
%let ftrlines=2;
data _null_;
file MyFile print header=hdr
  linesleft=remain;
set MyData end=last;
DATA _NULL_ statements...

if last or remain =< (&ftrlines+1) then
  link ftr;
return;

hdr:
header-creating statements...
return;
ftr:
addline=(remain - &ftrlines);
if addline > 0 then do i=1 to addline;
  put;
end;
put 'Report-Specific Footer Entry1';
put 'Report-Standard Footer Entry1';
if not last then put _page_;
return;
run;
```

Please note the operational line of code for this technique:
 if last or remain =< (&ftrlines+1) then
 link ftr;

This line can be placed after any put statement that begins a new report line.

TECHNIQUE III – NUMBERING PAGES WITH A TOTAL PAGE COUNT

Data reports most often have footers which contain page number entries. These page numbers may be listed along with a total page count for the report. It is relatively simple to update and list page numbers in a footer as a report is being built; however, including a total page count with page numbers can be tricky – the total number of pages may not be known until the report is completed.

If processing time is not an issue, a simple solution exists: create the report twice - once to determine the total page count, then a second time including this count with page numbers. This technique can be implemented with minimal effort; however, processing time does suffer. If long reports put too much of a strain on resources, then some other page numbering technique should be explored.

```
%let ftrlines=1;
%let pgcnt=;
%macro data_rpt;
data _null_;
length pgch $10 pgfmt $15;
retain pgnu 0;

DATA _NULL_ statements...
set MyData end=last;
More DATA _NULL_ statements...
return;
hdr:
header-creating statements...
return;
```

```
ftr:
footer statements...

* Update page numbers when *;
* a footer is inserted *;
pgnu=pgnu+1;
pgch=put (pgnu, 8.);
pgftr='Page '||pgch;
%if &pgcnt ne %then %do;
  pgftr=trim(pgftr)||" of &pgcnt";
%end;
%else %do;
  if last then call symput('pgcnt',pgch);
%end;

put @1 pgftr;
if not last then put _page_;
return;
run;

%mend data_rpt;
```

```
%data_rpt;
%data_rpt;
```

TECHNIQUE IV - THE HEADING RULER

Reports having entries listed in columns may require descriptive headings for each column of data. When programming this type of report, it is necessary to keep track of the position and length of each column heading - headings must not extend into one another; additionally, the length of the completed report heading must not surpass that set by the linesize option. The position of report entries must be accounted for also – they will be positioned under column headings and must not extend into one another as well. Keeping track of this information while programming can be challenging as well as time consuming. Yet this task can be simplified by the use of a heading ruler.

A heading ruler displays the number of character positions lengthwise available for a report. It can be referenced when creating column headers and when positioning report entries within a put statement. Programming reports becomes more straightforward when this information is readily available - guesswork about entry positioning is eliminated.

```
data _null_;
DATA _NULL_ statements...
put @1 fname @12 lname @23 gender;
return;
hdr:
put
  @1 '      Demographic Information'
  /@1
  /@1 'First      Last          '
  /@1 'Name       Name          Gender  ';
  /* 0123456789012345678901234567890 */
  /* 0          1          2          3 */
return;

ftr:
footer-creating statements...
return;

run;
```

TECHNIQUE V – THE DATA_NULL_ MACRO INTERFACE

Creating programs for a large collection of reports can be an arduous task, especially if separate program files are made for each report. Report programs created this way will be difficult to maintain - global changes will require considerable programming time. The time and effort involved with creating and maintaining report programs can be significantly reduced using the macro interface described below. This interface entails separating report inputs from DATA_NULL_ reporting programs. Report inputs are the defining characteristics of a report; they include information for headers, footers, and dataset variables present on a report. The DATA_NULL_ macro interface provides one method of organizing and storing report inputs. These inputs are captured by a DATA_NULL_ reporting program which then formulates the DATA_NULL_ statements needed to create a report. The number of DATA_NULL_ reporting programs required for a collection of reports depend on the degree reports differ from each other. If reports share a similar layout, then few DATA_NULL_ report programs are needed. If reports are varied and have little in common, then more DATA_NULL_ report programs will need to be created. In some cases, it may be possible to reduce the number of DATA_NULL_ report programs by defining report inputs which direct how a report is made.

Report inputs for a collection of reports may be stored in one file. Global changes are then much easier to make than if this information was spread out among many program files. Report programs are easier to maintain as well - one DATA_NULL_ report program can be used to create many reports thus reducing redundant code. The example below demonstrates how to use DATA_NULL_ report programs with report inputs stored in a macro interface.

```

/*****
/* File:          rpt_program.sas          */
/* Description:  DATA_NULL_ reporting */
/*              programs                  */
/* Contents:     c_report1                 */
/*****

/*****
/* Macro:        GetEntry                  */
/* Summary:      Returns the n-th entry   */
/*              of a list                  */
/* Parameters:   */
/* list - List of space-delimited        */
/* entries                                             */
/* pos  - Specifies the n-th entry        */
/*              to return                    */
/*****
%macro getentry(list,pos);
  %let entry=%scan(&list,&pos,%str( ));
  &entry
%mend getentry;

/*****
/* Macro:        Words                    */
/* Summary:      Counts the number of    */
/*              words in a string        */
/* Parameters:   */
/* string - The text string              */
/*****

```

```

%macro words(string);
  %local count words;
  %let count=1;
  %let word=%qscan(&string,
                  &count,
                  %str( ));
  %do %while(&word ne);
    %let count=%eval(&count+1);
    %let word=%qscan(&string,
                    &count,
                    %str( ));
  %end;
  %eval(&count-1)
%mend words;

/*****
/* Macro:        c_report1                */
/* Summary:      DATA_NULL_ report      */
/*              macro called to create  */
/*              a report                  */
/* Parameters:   */
/* indsn        - Dataset in the work    */
/*              library containing      */
/*              variables to be        */
/*              entered into the        */
/*              report                  */
/* outfile      - File name of the      */
/*              report created          */
/* hdr1-hdr10   - Report headers,       */
/*              enclosed in quotes      */
/* ftr1-ftr10   - Report footers,       */
/*              enclosed in quotes      */
/* lvar         - List of variables      */
/*              to be entered into      */
/*              the report              */
/* lpos         - List of variable      */
/*              positions in the        */
/*              report                  */
/*****
%macro c_report1(indsn=,
                outfile=,
                lvar=,
                lpos=,
                hdr1=,hdr2=,hdr3=,hdr4=,hdr5=,
                hdr6=,hdr7=,hdr8=,hdr9=,hdr10=,
                ftr1=,ftr2=,ftr3=,ftr4=,ftr5=,
                ftr6=,ftr7=,ftr8=,ftr9=,ftr10=);
  %local i nhdr nfttr;
  /* Count the number of */
  /* headers and footers */
  %let nhdr=0;
  %let nfttr=0;
  %do i=1 %to 10;
    %if %length(&&hdr&i) > 0 %then %let
      nhdr=&i;
    %if %length(&&ftr&i) > 0 %then %let
      nfttr=&i;
  %end;

  data _null_;
  file &outfile print header=hdr
    linesleft=remain;

  set &indsn end=last;

  *Formulate the put statement;
  put
  %do i= 1 %to %words(&lvar);
    @%getentry(&lpos, &i)
    %getentry(&lvar, &i)
  %end;;

```

```

    if last or (remain+1) =< &nftr then
    link ftr;
return;
hdr:
%do i=1 %to &nhdr;
    put &&hdr&i;
%end;
return;

ftr:
addline=(remain - &nftr);
if addline > 0 then do i=1 to
    addline;
    put;
end;
%do i=1 %to &nftr;
    put &&ftr&i;
%end;
if not last then put _page_;
return;
run;

```

```
%mend c_report1;
```

```

/*****
/* File:          rpt_input.sas          */
/* Description:   The defining          */
/*                characteristics for    */
/*                each report. Macro    */
/*                calls in this file    */
/*                can be executed to    */
/*                create reports        */
/*****
%c_report1(
  indsn=MyData,
  outfile=MyFile1,
  hdr1='          Demographics          ',
  hdr2='-----',
  hdr3='Last      First                    ',
  hdr4='Name      Name      Gender        ',
  /* 0123456789012345678901234567890 */
  /* 0      1      2      3 */
  lvar= lname   fname   gender,
  lpos= 1      10      20,
  ftr1='-----');

%c_report1(
  indsn=MyData,
  outfile=MyFile2,
  hdr1='          Zip Location          ',
  hdr2='-----',
  hdr3='Last      First      Zip          ',
  hdr4='Name      Name      Code*        ',
  /* 0123456789012345678901234567890 */
  /* 0      1      2      3 */
  lvar= lname   fname   zip,
  lpos= 1      10      20,
  ftr1='-----',
  ftr2='*9-Digit zip code used);

```

CONCLUSION

The five techniques presented in this paper can be used to make DATA_NULL_report programming quicker and easier. Each technique can be used on its own or in conjunction with other techniques. The strength of these techniques becomes evident when programming and maintaining a large collection of data reports.

REFERENCES

SAS Guide to Macro Processing, Version 6, Second Edition, SAS Institute, Inc.

CONTACT INFORMATION

Tyler Cole
 Pacific Data Designs
 900 NorthPoint Street, Suite C180
 San Francisco, CA 94109
 (415) 776-0660
 tcole@pdd.net



TRADEMARK INFORMATION

SAS and SAS Certified Professional are registered trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.