Paper 40

Code or DATA?

Ian Whitlock, Westat

Abstract

A Report Writing Seminar given at Westat was charged with finding new techniques to control complex DATA _NULL_ reports. This paper presents one of the techniques developed. DATA _NULL_ reports often require numerous tedious conditional PUT statements writing a mixture of literal information with data. In such cases one can view the report as a text input file with variable references, where one wants to conditionally print certain lines after making substitutions for variable references and conditionally inserting lines which are not in the input text file. This approach dramatically changes the way one can look at some DATA _NULL_ reporting problems.

A macro, COPYREPL, is used to generate DATA step code which automates the replacement of variable references with their values in a manner reminiscent of a mail-merge program. However, one quickly discovers the need to develop various control procedures to enhance the use of this macro. The talk will consist of presenting several examples ranging from simple to complex using this technique, concentrating on how to implement the control procedures required to use the macro in complex situations, and the macro.

Introduction

To put the matter starkly, consider two programmers asked to write a DATA _NULL_ program to present two columns of numbers. One chooses code to make the numbers part of his code.

```
DATA _NULL_ ;
   PUT @5 '    1' @20 '    2' ;
   PUT @5 '    2' @20 '    3' ;
   PUT @5 '   33' @30 '   -7' ;
   ....
RUN ;
```

The other chooses to make them data.

```
DATA _NULL_ ;
   INPUT X Y ;
   IF Y >= 0 THEN
      PUT @5 X 5. @20 Y 5. ;
   ELSE
      PUT @5 X 5. @30 Y 5. ;
CARDS ;
1 2
2 3
33 -7
...
RUN ;
```

In spite of the logical simplicity of the first program, most programmers will intuitively agree that the second program is far superior. Why? One way to judge the quality of a program is to consider making modifications and look at how hard it is to implement them. Here are some possible modifications:

- Insert a new row
- Accommodate seven digit numbers
- Skip a line every 5 rows
- Add a total for the first column
- Place an asterisk after every number over 10,000

The first suggestion is easy to implement with either program. Otherwise, implementation of the suggestions is far easier in the second program, particularly when the lists of numbers is long. The second program is better because it is more malleable. The first program is not malleable because the numbers have been placed in the code where one cannot get at them. In the second program the changes are trivial because they all involve simple manipulations of data.

In general programming languages are far better at manipulating data than at allowing easy changes to the code. Consequently, usually moving code to data increases flexibility, but at the expense of logical simplicity.

Typical DATA _NULL_ reports involve writing text and data in PUT statements. The question is, "Should the text be code or data?" Most often one finds the text placed in the program as code. In many cases the amount of information is minimal and it doesn't really matter, but in some the text may dominate the nature of the program as it did in the example above. In these cases it is worth considering making the text data, since the shift in view can dramatically simplify some reports.

The Macro COPYREPL

The standard argument given against the above suggestion is that the text is not fixed. There are parts that change depending on the data. For example consider a program to write short instructions to be followed for each patient.

```
Obtain sample of <sample> from patient
<patname> and test with the <color>
test kit dated <date>.  Enter the
results in <booklet> and return to
<collect>.
```

In the style of the first programmer one might write

```
put 'Obtain sample of ' sample
    'from' ;
put 'patient ' patname 'and test with'
;
....
```

The alternative is to find a way of accepting the preceding specification as data. One problem is, that in any given case it doesn't appear to be worth the effort. Now let's consider a macro COPYREPL to help with the problem. It's purpose is to replace array references indicated by 'nnn' with the values of the indicated variables on a line by line basis. Here is a call for the above problem:

```
%copyrepl ( line = line ,
            array = v ,
            text = rel ,
            varid = $,
            never = ~ )
```

where

- LINE indicates variable holding a line of text
- ARRAY names the array of values to use for replacement values
- TEXT indicates whether the surrounding text is fixed in position or should flow relative to the substituted values
- VARID names the surrounding symbol to identify variable references
- NEVER names symbol that is never used

Each variable reference above has been replaced by 'nnn' where nnn is an index to the array V which holds the various character string variables to be substituted. Here is the complete program.

```
DATA _NULL_ ;

   SET MAIN_DATA_SET ;
   ARRAY V (6) SAMPLE PATNAME COLOR
               DATE BOOKLET SUPER ;

   DO PTR = 1 TO 4 ;
     SET FORM_DATA_SET  POINT=PTR ;
     %COPYREPL ( ARRAY = V )
     LEN = LENGTH ( LINE ) ;
     PUT LINE $VARYING LEN ;
   END ;

RUN ;
```

In this example the form is read over and over. In the remaining examples we store the form as a character array for efficiency reasons.

A Pretty Box Example

The client insisted on camera ready reports. An example form for such a table is shown in the Appendix (A Pretty Box Example - Form). As shown the values for each column, nnn, have been replaced by the corresponding reference, nnn.

Part of the original program to produce these results is shown in the Appendix (A Pretty Box Example - Code). Here is a classic case for the code versus data problem. Note that most of the programming effort in this program is simply getting the pretty lines in the right place. There are 6 columns of figures to report. How easy would it be to add a new column? It is a nightmare of changes throughout the program.

Now we need a program to read the form, locate array references beginning with '$', read a record of data values and then make the appropriate replacements. What about adding a column? Just edit the form and provide for the new array element.

Here is the complete program for the form as shown.

```
title
  "Patient Demographics By Gender";
title2 "(Data As Of &SYSDATE)" ;

options ps=65 ls=175 nodate
        nonumber ;
libname dem 'c:\proj\data' ;

data _null_ ;
   array v (8) $ 8 _temporary_ ;
```

```
   infile 'c:\proj\tables\dem.dat'
          lrecl = 175 pad ;
   input  line $char175. ;
   if index ( line , '$' ) then
      link getdata ;
   %copyrepl (text=fixed, array=v);
   FILE PRINT ;
   put line $char175. ;
return ;

getdata:
   set dem.pdem ;
   v(1) = put ( col2 , 6. ) ;
   v(2) = put ( p1 , 8.2 ) ;
   v(3) = put ( col3 , 6. ) ;
   v(4) = put ( p2 , 8.2 ) ;
   v(5) = put ( col1 , 6. ) ;
   v(6) = put ( p3 , 8.2 ) ;
return ;
run ;
```

Here we see the same dramatic reduction in code as in the first example, accepting the macro COPYREPL as a tool and not part of the code. The logic is a bit strange because the main data file is now read in a subroutine conditionally invoked rather than the usual read on every iteration of the DATA step The conversion of the data into a character array is also outside the ordinary. However the code is actually quite simple and easy to modify after one realizes that the report structure drives the DATA step instead of the data values reported. Compare adding another column or changing the precision of the percents to the report using this program versus the one indicated in the appendix.

This should be enough to convince one that it is worth seeing how far we can push this idea of exchanging PUT code with an input text file.

One Observation Corresponds to One Form

Here is the classic program structure for a simple report requiring only substitution where one form is produced for each observation of data. For example one might think of sending sampling instructions to each school in an education survey.

```
data _null_ ;
   array text (&nl) $ 120
            _temporary_ ;
   array v (&nv) $ variable1
                   variable2 etc. ;

/* --- initialize text array --- */
   if _n_ = 1 then
   do i = 1 to dim ( text )
               until ( eof ) ;
      set form_data_set ;
      text ( i ) = line ;
    end ;

   set main_data_set ;

/* -------- write form -------- */
   file print ;
   do i = 1 to dim ( text ) ;
      line = text ( i ) ;
      %copyrepl ( array = v ) ;
      len = length ( line ) ;
      put line $varying. len ;
   end ;

run ;
```

Flexibility

As given above the DATA step doesn't have sufficient flexibility to handle many common situations. Here is a list of reasons various programmers have given why they couldn't use the copy and replace idea..

1. Sometimes I want fixed replacement, sometimes relative.
2. Sometimes some of the text should be deleted.
3. Sometimes I want one set of lines, sometimes another set.
4. I want numbered paragraphs.
5. I have to put a variable length list of numbers in the report (i.e. do something special that cannot be handled by a form).
6. I want this paragraph on a new page if it will not fit on the current page.

The thing that is missing here is control data to specify what should be done with each line. The answer to all of these problems is to provide control data with the form. In general the control data is stored in arrays parallel to the array holding the form. Typically subroutines will are used to keep the DATA step manageable.

Fixed Versus Relative

The pretty box example required fixed replacement so that the lines of the form stayed in place. This is typical of programs that use COPYREPL to write header information or a fixed format report. On the other hand, many situations require relative replacement. In other words, the text after the variable reference is allowed to float relative to the length of the value replacing the reference. Can one have it both ways in one program? Sure, let's make a type column to appear on each line before the form begins. The values could be 'F' for fixed and ' ' for relative. Now move the macro invocation out of the main DO-loop into a subroutine which decides how to call COPYREPL. In the code below, COPYREPL is invoked twice so that the question is turned into which block of code to execute instead of how to invoke the macro.

```
.....
array text (&nl) $ 120
           _temporary_ ;
array type ( &nl ) _temporary_ ;
.....

file print ;
do i = 1 to dim ( text ) ;
   link copyrepl ;
   put line ;
end ;
return ;

copyrepl:
   line = text ( i ) ;
   if type ( i ) = 'F' then do ;
      %copyrepl ( text = fixed ,
                  array = v )
   end ;
   else do ;
      %copyrepl ( array = v )
   end ;
return ;
```

Skipping Text

To indicate skipping text, let's add another control column before the form and a corresponding array CNTL to handle deletions. Let's let the values identify the condition, for example 'S_1', then you can test - is CNTL (I) = 'S_1' and is the S_1 condition fails then skip. In practice it is easier to say what is wanted rather than what is not. I have set a variable WANTED to 0 (false). The line then passes through a sequence of tests, if any test passes then WANTED is set to 1 (true).

```
.....
array text (&nl) $ 120
           _temporary_ ;
array type ( &nl ) _temporary_ ;
array cntl (&nl) $ 3 _temporary_ ;
.....

do i = 1 to &nl ;
   wanted = 0 ;
   if cntl = ' ' then wanted = 1 ;
   else
   if cntl ( i ) = 'S_1' and
      condition then
          wanted = 1 ;

   if wanted then
   do ;
      link copyrepl ;
      put line ;
   end ;
end ;
return ;
```

To handle either/or situations list them both and choose which is wanted for the current situation.

Numbered Paragraphs

Relative paragraph numbers are easily handled. Just assign a variable reference, typically '1, to hold the last used paragraph number. Then provide a means of incrementing this character variable.

```
.....
retain num '  0' ;

file print ;
do i = 1 to &nl ;
   if index ( text( i ), '$1$' )
   then num =
        put (input(num,3.)+1, 3.) ;
   link copyrepl ;
   put line ;
end ;
return ;
```

Special Problems

Some problems cannot be handled as input text. For example, suppose a list of names must be read from an auxiliary file and written on some given number of lines.

The CNTL array is handy for this purpose. All we need are a set of special values to cover the special problems. In fact one might view skipping as a special problem where nothing is to be done. Here is code to illustrate.

```
.....
array cntl (&nl) $ 3 _temporary_ ;
.....
do i = 1 to &nl ;
   if cntl ( i ) = 'MKL' and
      mkl = 1 then
         link mklist ;
   else
   do ;
      link copyrepl ;
      put line ;
   end ;
end ;
return ;

mklist :
   do until ( done = 1 ) ;
      set names (keep=name done) ;
      put @10 name $char20. ;
   end ;
   put ;
return ;
```

MKL is a variable on the main file which indicates whether a list is needed for this record. The value 'MKL' in the control column of the text file indicates where the special processing is to be done. The variable DONE on the names file is used to indicate the last name in each block of names.

Paging

Paging is often a problem in DATA _NULL_ reports when one wants to keep a whole paragraph together. Let's add another array PCNT giving the size of each paragraph on the first line of the paragraph. We now are in a position to make this kind of paging judgment.

```
   .....
   array pcnt (&nl) _temporary_ ;
   file print n = 60 ll = ll ;
   .....
   do i = 1 to &nl ;
      %copyrepl ( array = v ) ;
      if ll < pcnt ( i ) or
         cntl = 'NPG' then
            put _page_ ;
      put line ;
      end ;
   return ;
```

A Program Shell

Now let's put it all together. What sort of program can one expect using the tools introduced above?

```
data _null_ ;
   retain num '  0' ;
          /* replacement values */
   array v (&nv) $ 20 _temporary_;
  /* replacement type 'F', blank */
   array typ (&nl) $1 _temporary_;
            /* #paragraph lines */
   array pcntl (&nl) _temporary_ ;
   /* control array skip or sub */
   array cntl (&nl) $3 _temporary_
              /* the form array */
   array text (&nl) $ 200
                    _temporary_ ;

  /* load control & text arrays */
   if _n_ = 1 then link fill ;

   set main_data_set ;
   /* set up reference values */
   link setup ;

   do i = 1 to &nl; /* main loop*/
     oktoprnt = 0 ;
     /* decisions */
     select ( cntl ( i ) ) ;
       when ( ' ' ) oktoprnt = 1 ;
             /* usual conditions */
       when ( 'S_1' )
         if condition1 then
            oktoprnt = 1 ;
       when ( 'S_2' )
         if condition2 then do ;
           link s2prep ;
           oktoprnt = 1 ;
         end ;
       ......
                 /* special line */
       when ( 'R_1' )
         link routine1;
                 /* special list */
       when ( 'R_2' )
         link routine2 ;
       ......
       otherwise error
       'Bad cntl code ' cntl( i )=;
     end ;
```

```
     if oktoprnt then
     do ;
       if index (text( i ),'$1$')
         then num =
           put(input(num,3.)+1,3.);
       line = text ( i ) ;
       type = typ ( i ) ;
       needlin = pcntl ( i ) ;
       link copyrepl ;
       link printit ;
     end ;
   end ;          /* main loop */
return ;

/* common place to print line */
/* requires line and needlin  */
printit:
   file out n=60 ll=ll
            header=header ;
   if ll < needlin then
      put _page_ ;
   len = length ( line ) ;
   put line $varying. len ;
return ;

/* top of page routine called */
/* by SAS super or put _page_ */
header:
   pageno + 1 ;
return ;

fill:       /* fill text array */
   do f = 1 to &nl until ( eof ) ;
     set form_data_set ;
     pcntl ( i ) = needlin ;
     typ ( i ) = type ;
     cntl ( i ) = control ;
     text ( i ) = line ;
   end ;
   if i = &nl and not eof then abort ;
return ;

/* basic copy & replace routine */
/* requires line and type       */
copyrepl:
   if type = 'F' then
   do ;
     %copyrepl (text=fixed,array=v)
   end ;
   else
   do ;
     %copyrepl ( array = v )
   end ;
return ;

/* -------- user code -------- */

setup:    /* fill value array */
   .....
return ;

routine1:
   .....
   line = ..... ;
   oktoprnt = 1 ;
return ;

routine2:
   /* supervisor will decide */
   needlin = 0;
   do x = 1 to lastx ;
   .....
     line = ..... ;
     link printit ;
   end ;
return ;

s2prep:
.....
return ;
```

Note that the subroutines except for COPYREPL have been written so that they know nothing about the basic arrays. This means that they may be called from any where by anybody who sets up the needed parameters. This design is important because it means temporary changes can be made without fear of harming the basic array structure and it means they can be ported to other programs.

The basic routines should probably be made macros with parameters to indicate the variable names. In this case they are truly portable to other programs.

Can the TEXT array be completely blank? Yes! In this case the report is completely constructed by special subroutines. The order of calling these special subroutines is determined by CNTL and whatever testing is built into the shell.

Conclusion:

The macro COPYREPL makes the copy and replace form of DATA _NULL_ report easy to implement. Once one recognizes the need for control information and how to use it the program can be very flexible. This technique can be particularly important when the basic report

changes each time the program is run since the client can be taught to produce the form or at least how to specify the form so that it is easy typing for the programmer.

The author can be contacted by mail at:

Westat Inc.
1650 Research Boulevard
Rockville, MD 20850-3129

or by e-mail at:

whitloi1@westat.com

Appendix

A Pretty Box Example - Code

```
LIBNAME TEST 'C:\PROJ\DATA';
OPTIONS PS=65 LS=175 NODATE NONUMBER;

TITLE 'PATIENT DEMOGRAPHICS BY GENDER';

PROC FORMAT;
  VALUE $ DATFMT
     'N1'='13 - 18'
     'N2'='19 - 29'
     'N3'='30 - 39'
     'N4'='40 - 49'
     'N5'='50+    '
     'N6'='TOTAL PATIENTS'
     'N7'='White, Not Hispanic'
     ......;
  VALUE $ HDR
      'N1'='Age (Years)'
      'N7'='Race/Ethnicity'
      'N15'='HIV Risk Factors' ;
run;

%macro hbar;
    @34  '•' @80  '•' @91  '•' @102 '•'
    @113 '•' @124 '•' @135 '•' @147 '•'
%mend hbar;

%macro closebox;
    @34  '•' @80  '•' @91  '•' @102 '•'
    @113 '•' @124 '•' @135 '•' @147 '•'
%mend closebox;

%macro column;
    @34  '•' @80  '•' @91  '•' @102 '•'
    @113 '•' @124 '•' @135 '•' @147 '•'
%mend column;

TITLE2 "(DATA AS OF &SYSDATE)";

DATA _NULL_;
  SET TEST.PDEM END=EOF;
  FILE PRINT HEADER=NEWPAGE;

  IF _NAME_='N1' THEN PUT
    %hbar @36 _NAME_ $HDR. OVERPRINT
    @36 '••••••••••••' %hbar / %hbar ;

  IF _NAME_='N7' THEN PUT  overprint
    @35 112*'•' %column /
    %hbar @36 _NAME_ $HDR. OVERPRINT
    @36 '••••••••••••••' %hbar / %hbar;

  IF _NAME_='N15' THEN PUT overprint
    @35 112*'Ä' %column /
    %hbar @36 _NAME_ $HDR. OVERPRINT
    @36 '•••••••••••••••' %hbar / %hbar;

  IF _NAME_ IN ('N1','N2','N3','N4','N5')
  THEN DO;
    PUT  %hbar @38 _NAME_  $ DATFMT.
       @81 COL2 6.0 @92 P1 8.2 @103 COL3 6.0
       @114 P2 8.2 @125 COL1 6.0 @136 P3 8.2;
    RETURN;
  END;

  IF _NAME_='N6' THEN DO;
    put  %hbar  @84 '---' @106 '---'
       @128 '---' %hbar / %hbar
       @38 _NAME_  $ DATFMT. @81 COL2 6.0
       @92 P1 8.2 @103 COL3 6.0 @114 P2 8.2
       @125 COL1 6.0 @136 P3 8.2 ;
    RETURN;
  END;

  .......

RETURN;

NEWPAGE:
  PUT
    /// overprint @35 112*'•'
        @34  '•' @80  '•'
        @124 '•' @147 '•'
    /   @34  '•' @80  '•'
        @94  '    GENDER     '
        @124 '•' @147 '•'
    / overprint @80 44*'•'
        @34  '•' @80 '•'  @102 '•'
        @124 '•' @147 '•'
    /   @34  '•' @80 '•'  @102 '•'
```

```
    @124 '•' @147 '•'
    @35  ' '
    @83  '      MALE     '
    @105 '     FEMALE    '
    @132 'TOTAL'
/ overprint @80 67*'•'
    @34  '•' @80  '•' @91  '•'
    @102 '•' @113 '•' @124 '•'
    @135 '•' @147 '•'
    ......;
return ;
RUN;
```

Pretty Box Example - Form

DESCRIPTION	GENDER					
	MALE		FEMALE		TOTAL	
	NUMBER	PERCENT	NUMBER	PERCENT	NUMBER	PERCENT
Age (Years)						
13 - 18	1	2	3	4	5	6
19 - 29	1	2	3	4	5	6
30 - 39	1	2	3	4	5	6
40 - 49	1	2	3	4	5	6
50+	1	2	3	4	5	6
TOTAL PATIENTS	1	2	3	4	5	6
Race/Ethnicity						
White, Not Hispanic	1	2	3	4	5	6
Black, Not Hispanic	1	2	3	4	5	6
White, Hispanic	1	2	3	4	5	6
Black, Hispanic	1	2	3	4	5	6
Asian/Pacific Islander	1	2	3	4	5	6
Aleut Eskimo/American Indian	1	2	3	4	5	6
Other	1	2	3	4	5	6
TOTAL PATIENTS	1	2	3	4	5	6
HIV Risk Factors						
Male Homosexual/Bisexual Contact(s)	1	2	3	4	5	6
Drug Use by Injection	1	2	3	4	5	6
Hemophilia/Coagulation Disorder	1	2	3	4	5	6
Heterosexual Contact: Sex-IV Drug User	1	2	3	4	5	6
Heterosexual Contact: Sex-Bisexual Male	1	2	3	4	5	6
Blood Transfusion	1	2	3	4	5	6
Other	1	2	3	4	5	6
Unknown	1	2	3	4	5	6
No HIV Risk Factors	1	2	3	4	5	6
2 or More HIV Risk Factors	1	2	3	4	5	6
TOTAL PATIENTS	1	2	3	4	5	6

The Macro %COPYREPL

```
%macro copyrepl
    ( line = line ,     /* input line declared by user                  */
      never = '~' ,     /* symbol that does not apperar in report       */
      varid = '$' ,     /* symbol for variable reference                */
                  /* e.g. $n$ where n is number of array elements below */
      array = ,         /* character array using references above       */
      informat = 12. ,  /* informat for variable reference array        */
      text = relative   /* fix(ed) or rel(ative)                        */
    ) ;
   %if %quote(&array) = %str() %then
   %do ;
      %put COPYREPL: ARRAY has null value, ;
      %put &line not transformed. ;
      %goto mexit ;
   %end ;

   %let text = %upcase ( %substr ( &text , 1 , 3 ) ) ;

   length __temp $ 200 ;
   drop __x            /* index into &line numeric index into &array */
       %if &text = REL %then __wx ;
       %else __vx ;    /* character index into &array                */
        __temp         /* building area for new &line                */
   ;

   %if &text = REL %then
   %do ;
      &line = translate ( &line , &never , ' ' ) ;
      __temp = ' ' ;
      __x = index ( &line , &varid ) ;
      do while ( __x > 0 ) ;
        if __x > 1 then
           __temp = trimn ( __temp ) || substr ( &line, 1, __x-1 ) ;
        &line = substr ( &line, __x+1 ) ;
        __wx = input ( scan ( &line, 1, &varid ), &informat );
        __temp = trimn ( __temp ) || &array ( __wx ) ;
        __x = index ( &line, &varid ) ;
        if __x < length ( &line ) then
           &line = substr ( &line , __x + 1 ) ;
        else
           &line = ' ' ;
        __x = index ( &line , &varid ) ;
      end ;
      __temp = trimn ( __temp ) || &line ;
      &line = translate ( __temp , ' ' , &never ) ;
   %end ;
```

```
   %else
   %do ;
      length __vx $ 3 ;
      __x = index ( &line, &varid ) ;
      do while ( __x > 0 ) ;
        if __x > 1 then
          __vx = scan ( &line, 2, &varid ) ;
        else
          __vx = scan ( &line, 1, &varid ) ;
        substr ( line, __x, length(__vx)+2 ) = ' ' ;
        __temp = &array ( input ( __vx , &informat ) ) ;
        substr(line, __x , length(__temp)) = trim ( __temp ) ;
        __x = index ( &line, &varid ) ;
      end ;
   %end ;
%mexit:
%mend copyrepl ;
```