

PROGRAMS THAT WRITE PROGRAMS: HOW TO FLATTEN SAS® DATA SETS AND OTHER SOLUTIONS TO EVERYDAY PROBLEMS

John Iwaniszek
Denise Svendsgaard Williamson
Hoffmann Research Associates, Inc.
Chapel Hill, North Carolina

Abstract

Some data transfer tasks require that data sets be in a flat format (e.g. ASCII, EBCDIC). Producing flat files from SAS data sets necessitates writing a program that formats the data according to the contents of the parent data set. If the data set is to be reconstituted into a SAS data set after the transfer, then a program must be written to read the flat file. Two applications are discussed which address this problem. One is a program which uses the output data set from PROC CONTENTS to write two programs: one to create a text file based on the original SAS data set and the other to recreate the SAS data set with formats and labels intact.

The second application to be discussed is a one which creates a SAS file layout based on a mainframe data base file layout. Producing SAS data sets from flat files usually requires that one manually translates a file layout into input statements to be read by SAS. However, when the file layout is provided in electronic format, this information can be read and input statements can be generated using SAS.

Introduction

This paper is about two SAS programs which evolved out of the everyday business need to create SAS data sets out of text files and to create text files out of SAS data sets. Both programs themselves write programs or parts of programs which accomplish these tasks. One program uses output from the SAS system's own PROC CONTENTS as the basis of its process, while the other program uses output from another data management system to derive a text file's layout and create a program for generating SAS data sets. The first program (parent) to be discussed is a program which generates two programs (children) that convert SAS data sets to text files and back again.

First Application: SAS Data Sets to Text to SAS Data Sets

The parent program named IOL_DIB.SAS uses the output data set produced by PROC CONTENTS to produce two children. One child uses a PUT statement in a data step to write a SAS data set to text form. The other child reconstitutes the original SAS data set from the text file the first program creates. The second child applies all the data set and variable labels and formats and informats that were present in the original data set. The PROC CONTENTS output data set contains nearly all the information necessary to define each data element sufficiently to represent them in text form. Below are the PROC CONTENTS output and the parent and children. Note that the parent program contains macro variable parameters that allow the user to specify input and output data set and program names, and other documentary features of the child programs and data set files.

PROC CONTENTS Output

PROC CONTENTS Output Data Set as Read By IOL_DIB.SAS									
L	N	M	T	L					
1	2	3	4	5	6	7	8	9	10
1	SAS	SESG01	Train Staff Times to Departure	SESG01V	1	8			
2	SAS	SESG01	Train Staff Times to Departure	SESG01V	1	8			
3	SAS	SESG01	Train Staff Times to Departure	SESG01V	2	6			
4	SAS	SESG01	Train Staff Times to Departure	SESG01V	2	1			
5	SAS	SESG01	Train Staff Times to Departure	SESG01V	2	1			
6	SAS	SESG01	Train Staff Times to Departure	SESG01V	2	1			
7	SAS	SESG01	Train Staff Times to Departure	SESG01V	2	6			
8	SAS	SESG01	Train Staff Times to Departure	SESG01V	1	8			
9	SAS	SESG01	Train Staff Times to Departure	SESG01V	1	5			
10	SAS	SESG01	Train Staff Times to Departure	SESG01V	2	5			
V	A	L	F	F	F	N	I	I	
1	2	3	4	5	6	7	8	9	10
1	1	Employee Number	5	2	0	0	1	52	67
2	1	Equip. - Check Ride	0	0	0	0	0	1	4
3	2	Post. - Check Ride	0	0	0	0	0	0	18
4	3	Rate Employee Number	0	0	0	0	0	1	20
5	4	Rate Type	0	0	0	0	0	0	19
6	7	Code	0	0	0	0	0	0	36
7	6	Completion Date	MDYY	8	0	MDYY	8	0	1
8	9	Total Time Pre-LOFT	TDC	5	0	TDC	5	0	1
9	8	Type	0	0	0	0	0	0	42

```

"date dbm.toset(label='<idsetlabel>');           "/
- infile "<idset.txt">" lraclis=racient misover pad;    "/"
- length                                           "/"
//
/* end of put */
run;
data _null_ ;
file "spname_d2b.sav" mod;
set one end=last;
length
    lang $4.
;
label
    lang='Length of Variable'
;
if type=1 then lang=trim(left(lang))||'|'.|';
if type=2 then lang=s'||(trim(left(lang))||'|'.|';

put
$17 name $25 lang;

if last=1 then
put
//
input
//
/* end of put */

run;
data _null_ ;
file "spname_d2b.sav" mod;
set one end=last;
length
    position 8.
    stepo $7.
    readfm $10.
    rctyp $1.
;
retain
    position 1
;
label
    position='Position of Variable on Record'
    stepo='Position With 8 concatenated in Front'
    readfm='Read Format'
    rctyp='Variable Type'
;

stepo='8'||(trim(left(position)));
if type=1 then rctyp='8';
else rctyp='';
if format="" then readfm=trim(left(format))||'(trim(left(length))||'|'.|';
if type=1 then length=inmleng;
if format="" then readfm=rctyp||(trim(left(length))||'|'.|';

put
$17 stepo $25 name $25 readfm;

if type=1 then position=position+inmleng;
if type=2 then position=position+length;

if last=1 then
put
format //
//

run;
data _null_ ;
file "spname_d2b.sav" mod;
set one end=last;
length
    outfm $10.
;
label
    outfm='Output Format'
;
if format="" then outfm=trim(left(format))||'(trim(left(format))||'|'.|')|'(trim(left(formatd)));
if format="" and format1=""=0 then do;
    outfm=trim(left(format))||'|'.|')|'(trim(left(formatd)));
end;
if format1=""=0 then put $17 name $27 outfm;

if last=1 then
put
//
inform //
//

run;
data _null_ ;
file "spname_d2b.sav" mod;
set one end=last;
length
    infm $10.
;
label
    infm='Input Format'
;
if informat="" then infm=trim(left(informat))||'(trim(left(inform))||'|'.|')|'(trim(left(informd)));
if informat="" and inform=""=0 then do;
    infm=trim(left(inform))||'|'.|')|'(trim(left(informd)));
end;
if informat=""=0 then put $17 name $27 infm;

if last=1 then
put
//
label //
//

run;
data _null_ ;
file "spname_d2b.sav" mod;
set one end=last;
label=compress(label,' ');
put
$17 name '=' ' ' label ' '
//
if last=1 then
put
//
//

run;
proc contents data=dbm.toset; run;
proc print data=dbm.toset(obs=25); run;

run;
```

Child #1

```
*ses_d2a.sas > This program creates ASCII data set from SAS data set.
SESUG Demo
Train Staff Times to Departure

input: sesugl.ssd
output: sesugl.txt

John Iwaniszek, JRI;
libname dmh '.';
options ps=60 ls=132;

data _null_;
  file "sesugl.txt" lrecl=99;
  set dmh.sesugl;
  put

      01      EMPNO
      017     EQ_RIDE
      023     POS_RIDE
      024     RATEMTYP
      025     RATEMPNO
      041     RIDEDT
      057     RIDECD
      063     RIDETYP
      068     RIDEHOUR
      084     EMPDIV
      ;;

run;
```

Child #2

```
*ses_d2b.sas > This program creates SAS Data set from ASCII data set.
SESUG Demo
Train Staff Times to Departure

input: sesugl.txt
output: sesugnew.ssd

John Iwaniszek, JRI;
libname dmh '.';
options ps=60 ls=132;

data dmh.sesugnew(label='Train Staff Times to Departure');

  infile "sesugl.txt" lrecl=99 missover pad;
  length
      EMPNO      8.
      EQ_RIDE     $6.
      POS_RIDE    $1.
      RATEMTYP    $1.
      RATEMPNO    8.
      RIDEDT      8.
      RIDECD      $6.
      RIDETYP     $5.
      RIDEHOUR    5.
      EMPDIV      8.
      ;;

  input
      01      EMPNO      16.
      017     EQ_RIDE    $6.
      023     POS_RIDE    $1.
      024     RATEMTYP    $1.
      025     RATEMPNO    16.
      041     RIDEDT     MMDDYY8.
      057     RIDECD      $6.
      063     RIDETYP     $5.
      068     RIDEHOUR    TIMES.
      084     EMPDIV      16.
      ;;

  format
      RIDEDT     MMDDYY8.0
      RIDEHOUR    TIMES.0
      EMPDIV     5.2
      ;;

  informat
      RIDEDT     MMDDYY8.0
      RIDEHOUR    TIMES.0
      ;;

  label
      EMPNO = 'Employee Number '
      EQ_RIDE = 'Equip. - Check Ride '
      POS_RIDE = 'Posl. - Check Ride '
      RATEMTYP = 'Rater Type '
      RATEMPNO = 'Rater Employee Number '
      RIDEDT = 'Completion Date '
      RIDECD = 'Code '
      RIDETYP = 'Type '
      RIDEHOUR = 'Total Time Pre-LOFT '
      EMPDIV = ' '
      ;;

run;
proc contents data=dmh.sesugnew; run;
proc print data=dmh.sesugnew(obs=25); run;
```

Word of Caution

The user should be cautioned that there is nothing inherent in the PROC CONTENTS output that exactly defines the maximum number of characters or places a numeric data element possesses. A numeric data element with length 8. may be one digit long or it may be 32 digits long. In order to allow for this ambiguity, IOL_DIB.SAS (the parent program) contains a parameter in the form of a macro variable that provides a way for the user to set the length of all unformatted numeric data elements. This parent can be set at the top of the program in the parameter section.

The second exception presents a more difficult problem and appears not to have as neat a solution as the first. The programs written by IOL_DIB.SAS do not preserve the true values of numeric data elements which were previously formatted in the data set. For example, the number 10.2366 may be formatted in a data step to conform to the format 6.3. Under this format 10.2366 will appear as 10.237.

If the user keeps these two issues in mind and also keeps an eye out for unforeseen problems, then the parent program will perform perfectly and may be a useful addition to the SAS user's toolbox.

Second Application: Text to SAS Data Set

Everyday business may require the need to read electronic text files into SAS data sets. Often these text files are produced from an application that can also produce an electronic representation of the data layout. In these instances, SAS can be used to create an INPUT and LABEL statement that can be pasted into a SAS data step and will form the foundation of a larger SAS program.

In order to write a program that uses an electronic layout file to write INPUT and LABEL statements, it is necessary to perform three general steps:

1. Study the layout file to determine SAS format equivalencies.
2. Identify other benchmarks, such as labels and variable names, in the layout file to be used by the program that produces the SAS code.
3. Edit the SAS code that results from the parent program. The data base application that produced

the data and layout files may have significant incompatibilities (long variable names, for example) that only a human can resolve efficiently. This final step is very important to verify that the program will work as expected.

Example

A file layout produced by a non-SAS data base application is shown below.

empno	9(06)	Employee Number
ln	X(25)	Last Name
fn	X(15)	First Name
sex	X	Sex Code
sec	X	EEO Code
birthdate	9(06)	Birth Date
age	99	Age
location	X(03)	Work Location
prorgcode	X	Primary Organization Code
prorgtitle	X(25)	Primary Organization Title
jc	X(05)	Job Code
time_in_job	9(3)V9	Time since effective date of job
rate	9(5)V99	Hourly pay
salary	9(7)V99	Rate * 2080
service	99V9	Length of time of Digital employment
prop_grd_move	999	Signed indicator of movement in pay grade

In the file layout above, an 'X' indicates a character variable, while a '9' indicates a numeric variable. The number in parentheses indicates the length of the variable for variables with length longer than 2. A 'V' indicates there is a decimal place for numeric variables, with the number of digits after the decimal indicated by '9's. In addition, there is a variable with a format 'S99' which is a numeric variable with a sign (+/-) preceding the number.

Also note that the variable name appears in the first column of the file layout. The variable names were transferred into the INPUT statement as they appeared in the file layout and edited later. The format of the variable begins in column 15, followed by a description of the variable in column 27.

The preceding analysis of the file layout was used to write the following program. This program writes a SAS program containing INPUT statements and LABEL statements.

Parent

```

/* CRINPUT.SAS
/* Create SAS Program to input data by reading file layout
/* Create labels for variables from variable descriptions
/* Input: LAYOUT.SAS
/* Output: INPUT.SAS

/* Enter Name of Program You Want to Create
%let pgm = INPUT;

/* Enter Name of File Layout
%let layout = layout.txt;

DATA layout(drop=laglen);

```

```

/* Read Record Layout */

/* determine starting position of variable */
retain start;
if _n_ = 1 then start = 1;

/* read file layout */
infile "layout";
input @ 1 variable $13.
      @15 type $ 1.
      @16 length $ 7.
      @27 desc $40.

;;;

/* determine type of variable */
if type = 'X' then char = 'S';
if type = '9' then char = 'N';
if type = 'S' then char = 'N';

/* determine whether format includes a parenthesis */
/* determine column position of closing parenthesis (if one exists) */
check=')';
whereck=index(length,check);
if whereck>0 then max=whereck;

/* determine length of variable */
if length=' ' then nlength=1;
else if length='9' or length='X' then nlength=2;
else if length='99' then nlength=3;

/* determine length of char variable if format includes a parenthesis */
/* this example assumes a maximum length of 99 */
if type='X' and
  substr(length,1,1)='(' then nlength=substr(length,2,max-2);

/* determine length of numeric variable */
else if type='9' then do;

/* determine whether there is a 'V' indicating a decimal value */
look='V';
where=index(length,look);

/* if no 'V' - data length from amount in parentheses */
if where=0 then do;
  if substr(length,1,1)='(' then nlength=substr(length,2,max-2);
end;

/* if 'V' then determine length of variable */
/* (in this example, ASCII data does not include '.') */
else if where>0 then do;
  /* if format does not include parentheses */
  if substr(length,1,1) ne '(' then do;
    nlength=length(substr(length,1,where-1)) + 1 +
      length(substr(length,where+1));
    dec=length(substr(length,where+1)); /* data length after decimal */
  end;

  /* if format does include parentheses */
  if substr(length,1,1) = '(' then do;
    nlength=substr(length,2,max-2) + length(substr(length,where+1));
    dec=length(substr(length,where+1));
  end;
end;

/* create character variable to contain length after decimal */
length cdec $1;
cdec=cdec;
if cdec='.' then cdec=' ';

/* determine starting position of variable */
laglen=lag(nlength);

if _n_ = 1 then output;
else do;
  start=start+laglen;
  output;
end;

run;

/* Output header for program */

data _null_;
  file "pgm..sas";
  put @1
    /* $5 "pgm"
    /*
    /*
    /*
    /* Input Data Set:
    /* Output Data Set:
    /*
    /*
    $1 **** Add Libname here;
    //
    $1 'DATA' ; //
    $3 'INFILE' ; //

  ;;
  return;
run;

/* Output INPUT statements */

data _null_;
  set layout nobs=lastobs;
  file "pgm..sas" mod;
  if _n_ = 1 then
    put @3 'INPUT @' @10 start 3. @14 variable @26 char @29 nlength 2.
      /* cdec @33 '/' ' desc @73 ' '//';
  else if _n_ = lastobs then
    put @3 '
    @' @10 start 3. @14 variable @26 char @29 nlength 2.
      /* cdec @33 '/' ' desc @73 ' '//';

```

```

else do;
  / 03 ';;;';
  put 03 ' ' 010 start 3. 014 variable 028 char 029 nlength 2.
  ' ' cdec 033 ' ' ' desc 073 ' ' /';
end;
return;
run;

/* Output LABEL statements */

data _null_;
set layout nobobs=lastobs;
file "spgm..sas" mod;
if _n_ = 1 then
  put /
  03 'LABEL' 010 variable 024 "-" 026 desc 065 "'";
else if _n_ = lastobs then
  put 03 ' ' 010 variable 024 "-" 026 desc 065 "'";
/ 03 ';;;';
else do;
  put 03 ' ' 010 variable 024 "-" 026 desc 065 "'";
end;
return;
run;

/* Output statements to provide contents of data */

data _null_;
file "spgm..sas" mod;
put /
01 'PROC CONTENTS;' / 01 'RUN;';
return;
run;

```

The program above produces the following program:

Child (before)

```

/* INPUT */
/*
/*
/*
/*
/* Input Data Set:
/* Output Data Set:
/*

*** Add Libname here:

DATA
;

INFILE
;

INPUT 0 1 empno 6. /* Employee Number
0 7 ln $25. /* Last Name
0 32 fn $15. /* First Name
0 47 sex $ 1. /* Sex Code
0 48 eeo $ 1. /* EEO Code
0 49 birthdate 6. /* Birth Date
0 55 age 2. /* Age
0 57 location $ 3. /* Work Location
0 60 prmorgcode $ 1. /* Primary Organization Code
0 61 prmorgtlt $25. /* Primary Organization Title
0 86 jc $ 5. /* Job Code
0 91 time_in_job 4.1/* Time since effective date of job
0 95 rate 7.2/* Hourly pay
0102 salary 9.2/* Rate * 2080
0111 service 3.1/* Length of time of employment
0114 prop_grd_move 3. /* Signed indicator of movement in pay g */

;;;

LABEL empno -'Employee Number
ln -'Last Name
fn -'First Name
sex -'Sex Code
eeo -'EEO Code
birthdate -'Birth Date
age -'Age
location -'Work Location
prmorgcode -'Primary Organization Code
prmorgtlt -'Primary Organization Title
jc -'Job Code
time_in_job -'Time since effective date of job
rate -'Hourly pay
salary -'Rate * 2080
service -'Length of time of employment
prop_grd_move -'Signed indicator of movement in pay grd'

;;;

PROC CONTENTS;

RUN;

```

The program above can be edited to include variable names of the proper length and any comments. The following program is the result of such editing.

Child (after)

```

/* Read data provided by XYZ Corporation */
/*
/*
/*
/* Input Data Set: xyz.dat
/* Output Data Set: hoffmann.ssd
/*

libname hra 'hra';
options ps=60 ls=132;

DATA hra.hoffmann;

INFILE 'xyz.dat' lrecl=130;

format bdate mddy6.;

INPUT 0 1 empno 6. /* Employee Number
0 7 ln $25. /* Last Name
0 32 fn $15. /* First Name
0 47 sex $ 1. /* Sex Code
0 48 eeo $ 1. /* EEO Code
0 49 bdate mddy6. /* Birth Date
0 55 age 2. /* Age
0 57 location $ 3. /* Work Location
0 60 prmorgcd $ 1. /* Primary Organization Code
0 61 prmorgtlt $25. /* Primary Organization Title
0 86 jc $ 5. /* Job Code
0 91 timeinjb 4.1/* Time since effective date of job
0 95 rate 7.2/* Hourly pay
0102 salary 9.2/* Rate * 2080
0111 service 3.1/* Length of time of employment
0114 propgdmv 3. /* Signed indicator of movement in pay g */

;;;

LABEL empno -'Employee Number
ln -'Last Name
fn -'First Name
sex -'Sex Code
eeo -'EEO Code
bdate -'Birth Date
age -'Age
location -'Work Location
prmorgcd -'Primary Organization Code
prmorgtlt -'Primary Organization Title
jc -'Job Code
timeinjb -'Time since effective date of job
rate -'Hourly pay
salary -'Rate * 2080
service -'Length of time of employment
propgdmv -'Signed indicator of movement in pay gra'

;;;

PROC CONTENTS;

RUN;

```

Conclusions

The process of translating a file layout into input statements to be read by SAS requires some time. It is therefore not efficient when the file layout is brief. However, for lengthy file layouts, it is particularly useful to have the input statements generated automatically. Also, this process allows for simple maintenance in the event the layout is changed.

The authors may be contacted at:

Hoffmann Research Associates, Inc.
 111 Providence Road
 Chapel Hill, NC 27514
 (919) 489-8000

SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA.