

Paper 175-29

Write SAS® Code to Generate Another SAS® Program A Dynamic Way to Get Your Data into SAS®

Linda Gau, Pro Unlimited @ Genentech, Inc., South San Francisco, CA

ABSTRACT

In this paper we introduce a dynamic way to create SAS data sets from raw data files (flat files) by using SAS®/Base only. You only need to write one SAS program, and it will handle all kinds of raw data – fixed length, delimited, free-formatted or even hierarchical files. You may also apply the same algorithms with little modification to output raw data files from SAS data sets.

INTRODUCTION

It is often the case that many raw data files with varying structure need to be read into SAS data sets for analysis using SAS software. Writing a SAS program to read each file results in many similar programs with minor variation due to different starting points, variable names, data types, etc. If your clients change a file structure, for example, from a fixed length file to a delimited file, or vice versa, you may need to re-write the SAS programs to read raw data. While these programs are technically trivial, writing and maintaining them can be an onerous, time-consuming task.

Wouldn't it be nice to write one program which will generate tons of SAS programs and handle any data files? Yes, you can do this by creating a data layout table to store the specification of the data files and using DATA _NULL_ to create data dependent SAS programs. Alternatively, by using a macro, you have better control of your source code, and do not need to output SAS programs to the host system.

READING RAW DATA


Manual Programming to Read Raw Data

You can use column or formatted input to read data in fixed columns or standard character and numeric data.

```
data compare;
  infile 'external-filename';

  input idum $ 1-4 name $ 5 - 16 depends 18;

  input idum $4. name $12. depends 1.;
```



You may also use formatted input to read nonstandard character and numeric data, or date values, and convert them to SAS date values by using Informats: \$CHARw., COMMAw., DATEw., MMDDYYw., etc.

You can use list input to read data that are not in a fixed location.

```
INFILE fileref DLM='character';
INPUT @column var1 $var2 ....;

data compare;
  infile 'external-filename' dlm = '09'X;
  input idum $ name $ depends;
```

The following is an example that we might deal with in our daily job:

```
filename in 'xxxxxxxxxx';

data library.file1;

infile in lrecl=529 recfm=f;

input @ 1 ssn $9.
```

```

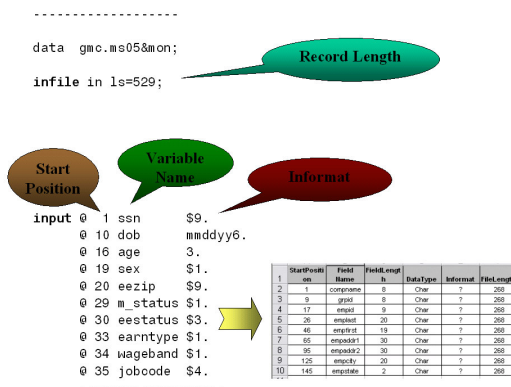
@ 10 dob      mmddy6.
@ 16 age      3.
@ 19 sex      $1.
@ 20 eezip    $9.
@ 29 m_status $1.
@ 30 eestatus $3.
@ 33 earntype $1.
@ 34 wageband $1.
@ 35 jobcode  $4.
@ 39 salary   zd11.2
@ 50 contserv mmddy6.
@ 56 pidays   5.2
@ 61 yearserv 2.
@ 63 pensionu $3.
@ 66 loca     $4.
@ 70 publ8    $7.
@ 77 ee_site  $8.
.....

```

The Dynamic Solution

Step 1: Create Data Layout Table

Let's dissect the code in the previous example. Basically, it can be broken down into four elements – record length, starting position for the variable, variable name, and information regarding data type and length of variable or informat.



You may store the information in a data file, let's name it "Data Layout Table". You may store the data layout table in a MS Excel spreadsheet (Fig. 1), then write a SAS program to read the data layout table into the SAS system, or enter the information directly into a SAS data set (Fig. 2).

	StartPositi	Field Name	FieldLengt	Data Type	Informat	File Length
1	on					
2	1	compname	8	Char	?	268
3	9	grpnd	8	Char	?	268
4	17	empnd	9	Char	?	268
5	26	emplast	20	Char	?	268
6	46	empfirst	19	Char	?	268
7	65	empaddr1	30	Char	?	268
8	95	empaddr2	30	Char	?	268
9	125	empcity	20	Char	?	268
10	145	empstate	2	Char	?	268

Fig. 1

The following program reads a data layout table as a tab delimited file into a SAS data set. Please note that there are various ways to input data, so if you prefer another solution, that's OK.

```

data tblay;

  length
    comp ftype $2 tbla_nam $8 tbla_len 4.
    tbla_dty $10 tbla_fmt $12;

  infile "&tabldir/tb_lay.txt"
    dlm='09'x missover dsd end=eof ;
  input
    comp $ ftype $ @;
  .....

  input

    tbla_pos $
    tbla_nam $
    tbla_len
    tbla_dty $
    tbla_fmt $
    tbla_fle $;
  .....

```

tbla_nam	tbla_len	tbla_dty	tbla_fmt	tbla_pos	tbla_fle
COMPNAM	8	CHAR		1	268
GRPID	8	CHAR		9	268
EMPID	9	CHAR		17	268
EMPLAST	20	CHAR		26	268
EMPFIRST	19	CHAR		46	268
EMPADDR	30	CHAR		65	268
EMPADDR	30	CHAR		95	268
EMPCITY	20	CHAR		125	268
EMPSTATE	2	CHAR		145	268
EMPZIP	10	CHAR		147	268
EMPAREA	5	CHAR		157	268
PHONE	8	CHAR		162	268
EMPSEX	1	CHAR		170	268
EMPDOB	8	DATE	CCYYMMDD	171	268
EMPJOBDA	8	DATE	CCYYMMDD	179	268
EMPJOBID	8	CHAR		187	268
EMPJOBTI	28	CHAR		195	268
ACTIVE	1	CHAR		223	268
SUPERNM	30	CHAR		224	268

Fig. 2

Step 2: Writing Dynamic Code to Read Raw Data -- Using DATA _NULL_

A lot of programmers use DATA _NULL_ for writing custom reports or raw data files. However, the same concept of writing a raw data file using FILE and PUT statements can be applied to write a SAS program. Instead of naming raw data files with an extension of .txt or .dat, you name it with an extension of **.sas**.

```

data _null_ ;
set tblay end=eof ;

file "&userdir/&compid&filetype&sysnum.lay.sas";

```

The SET statement simply tells SAS to read the data layout table, which stores the file structure specifications.

The next step is to write the code that you normally would for reading raw data, but using the a PUT statement and replacing the information for record length, starting position, variable names and informats with the fields in the data layout table.

```

if _n_ =1 then do;
put
"data XXXXX; "

/ 'infile "&datadir/&compid&filetype"          missover ls='  tbla_fle
';`

/ 'input'
;
end;

```

```

data onecomp ;
infile
"&datadir/&compid&filetype"
missover ls = 268 ;
input

```

Step 3: Recode Data Type and Informats

Data can be in different forms and the flat files may be generated by other software, e.g., Oracle, DB2, C, etc., so different keywords might be used to represent character or numeric data. The following are a few examples:

- Character: CHAR, VARCHAR2, etc.
- Numeric: NUMBER, INTEGER, SHORT, LONG, etc.

Thus, we need to do data cleaning on the data types first, and create informats to make our dynamic coding easier.

1. Data cleaning for character data:

```
if upcase(tbla_dty) =: 'VARCHAR'
then tbla_dty = 'CHAR      ';
```

2. Data cleaning for numeric data:

```
if upcase(left(tbla_dty)) in
( 'NUMBER      ', 'INTEGER   ')
then tbla_dty = 'NUMERIC   ';
```

3. Data cleaning for date variables:

```
if upcase(tbla_dty) =: 'DATE' then do;
  dtcnt = dtcnt +1;
  dts(dtcnt) = tbla_nam ;
  if upcase(left(tbla_fmt)) = 'CCYYMMDD'
  then tbla_fmt = 'yymmdd8.' ; else
  if upcase(left(tbla_fmt)) = 'MM/DD/CCYY'
  then tbla_fmt = 'mmdyy10.' ; else
  if upcase(left(tbla_fmt)) = 'MMDDYY10'
  then tbla_fmt='mmdyy10.' ; else
  if upcase(left(tbla_fmt)) = 'MMDDYY10.'
  then tbla_fmt='mmdyy10.' ; else
  if upcase(left(tbla_fmt)) = 'MMDDYY8'
  then tbla_fmt='mmdyy8.' ; else
  if upcase(left(tbla_fmt)) = 'MMDDYY8.'
  then tbla_fmt='mmdyy8.' ; else
  if upcase(tbla_fmt) = " "
  then tbla_fmt='mmdyy8.' ;
end;
```

In the meantime, you might also need to recode informats if necessary:

```
if upcase(tbla_dty) =: 'CHAR' then do;
  if tbla_fmt = " " then tbla_fmt = '$' ||
    compress(tbla_len)|| '.' ;

  else tbla_fmt = tbla_fmt;
end;

if upcase(tbla_dty) =: 'NUMERIC' then do;
  if tbla_fmt = " " then
    tbla_fmt = compress(floor(tbla_len)
    )||'.' ;
  else tbla_fmt = tbla_fmt;
end;
```

All the tedious work is done. Now comes the fun part, dynamic coding, and you will see how great and powerful a solution it is.

Usually, you will write the starting point, SAS variable name and informats for each variable that you are going to read into a SAS data set. Unfortunately, if a file with hundreds of variables needs to be read as a SAS data set, you need to write hundreds of lines by the following programming format.

01	COMPNAME	\$8.
09	GRPID	\$8.
017	EMPID	\$9.
026	EMPLAST	\$20.
046	EMPFIRST	\$19.
065	EMPADDR1	\$30.
095	EMPADDR2	\$30.
0125	EMPCITY	\$20.
0145	EMPSTATE	\$2.

Since we stored the data specification in the data layout table, we will replace the hundreds of lines of code with the following four lines of code:

```
if tbla_pos ne . then do;
    put @10 '@' tbla_pos +1
        tbla_nam +1 tbla_fmt ;
end;
```

At the end of the program for reading a raw data file, you will format some non-standard data, e.g. date variables, so it will make sense to the general users. One tip here is that you can store the date variables into an array, so you can dynamically format those date variables into the formats you prefer.

```
if eof then do;
    put ';'
    / 'format' ;

    do i=1 to dtcnt ;
        put dts(i)
    end ;

    put "mmddy10." /
    '; run; ' ;
end;
```

```
;
format EMPDOB
      EMPJOBDA
      mmddy10.;
run;
```

Step 4: Run Program that Actually Read Raw data

Finally, you may execute the program for reading raw data into a SAS data set by using %include statement.

```
%INCLUDE "&USERDIR/&COMPID&FILETYPE&SYSNUM.LAY.SAS";
```

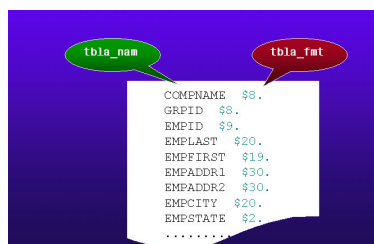
READ DELIMITED FILES

To further extend the program's functionality, consider delimited files. With a little bit of modification to the previous program, you can also read delimited files by using DELIMITER=, or DLM=, option in the INFILE statement.

```
if _n_ = 1 then do;
    put "data XXXXX; " /;
    put "length comp $2 " @;
    ...

    put 'infile "&datadir/&compid&filetype"
        dlm = "09"x dsd missover; ' /;
    put 'input' /;
end;
```

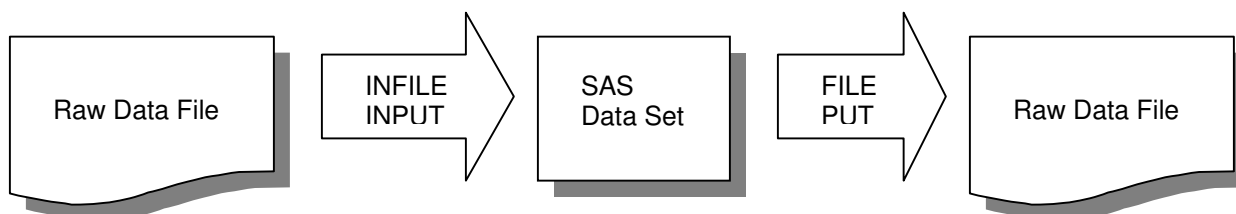
As we described before in the section on reading fixed length files, we may also generate data dependent SAS programs using the information stored in the data layout table.



```
PUT @10 TBLA_NAM +1 TBLA_FMT ;
```

WRITING RAW DATA

In addition to reading raw data into SAS data sets, you may modify this program to write flat files. Since it uses the same method as we just described but using FILE and PUT statements, we won't illustrate it here again.



Here is the code:

```

data _null_ ;
  file "&userdir/&compid&filetype&sysnum.asc.sas";
  set tbnorm end=eof ;
  .....

  if _n_ =1 then do;
    put "data _null_ ; "
      / 'options missing = " " ;'
      / "set outdir.&compid&filetype&dt ;"
      / 'file
        "&textdir/&compid&filetype&dt&sysnum..txt"
          ls=' tbnm_file ' ;'
      / 'put'
      ;
    end ;

    .....

    put @10 '@' tbnm_pos +1 tbnm_nam +1
      tbnm_fmt ;

    .....

    if eof then do;
      put ';'
      / ' run; ' ;
    end;

    %include "&userdir/&compid&filetype&sysnum.asc.sas";

```

ALTERNATIVE CODING – MACRO SOLUTION

There are various ways to accomplish this solution. If you are a macro guru, you might like to write dynamic code using SAS macro language. Here is a solution provided by Brit Harvey. We are just providing the idea, the rest of the

programming is up to you.

```

%macro ReadFile (
    MetaFile =
    /* INPUT METADATA FILE */
    , LineSize =
    /* LINESIZE FROM CLIENT */
    , DataFile =
    /* INPUT DATA FILE */
    , Out =
    /* OUTPUT DATASET */
);

%local ReadCode ;
/* THIS IS A MACRO VARIABLE THAT WILL CONTAIN CODE TO READ DATA FILE. */

/* WRITE THE DATA, INFILE AND INPUT STATEMENTS */
%let ReadCode = data &out %str(;) infile = &DataFile missover
ls=&LineSize %str(;) input ;

%do until end of &MetaFile;

    Read a line of &MetaFile.
    Do conversion of date, char, and numeric informats.
    Generate the input statement line, (call it &inline).

    %let ReadCode = &ReadCode &inline ;
%end ;

    Generate the formats (call them &formats).

%let ReadCode = %str(;) &ReadCode &formats %str(;) run %str(;) ;

/* OPTIONALLY PUT THE GENERATED CODE TO THE LOG FOR DEBUGGING */
%put Review generated code ;
%put &ReadCode ;

/* EXECUTE THE GENERATED CODE TO READ THE FILE */
&ReadCode

%mend ReadFile ;

```

CONCLUSIONS

I hope this paper will provide you with a great solution and make your work much easier if you face the same situation as I did. In the future, it might also trigger you to write a paper to simplify some routine work and make the programmer's life easier.

ACKNOWLEDGMENTS

I would like to express my appreciation to Brit Harvey for his helpful suggestions in the section of "Alternative Coding – Macro Solution", and Alan Nakamoto for editing my paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Linda Gau
 Pro Unlimited @ Genentech, Inc.
 1 DNA Way, Mail stop # 59
 South San Francisco, CA 94080
 (650) 225-8556
 Email: gau.linda@gene.com
 Email: csgau@yahoo.com

Please note that all the SAS programs in this paper were written for the UNIX platform, if you want to run these examples in other platforms, you may need to modify some SAS statements.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.