MWSUG 2019 - Paper PO-028

Generating SAS Datasets from ASCII Files Using a Crosswalk

Jose Centeno, NORC at the University of Chicago, Chicago, IL

ABSTRACT

In real-life applications, it is common to have a corresponding crosswalk in order to read raw files into SAS, especially if you're dealing with numerous output files and hundreds of variables with particular formats attached. In many cases, you will find yourself with the task of removing, adding, and/or updating variables which could become challenging or tedious. In this paper we will describe how with the help of a few macros we can reduce this effort as well as greatly decrease the number of lines of code in your main program. This program will be easier to maintain, less error-prone and will be easily deployed for other projects.

This paper assumes a basic understanding of SAS data step programming, and a basic understanding of SAS Macros.

INTRODUCTION

One of the projects I am currently working on involves reading and maintaining approximately 90 ASCII files that are ultimately converted into SAS datasets. Originally, these were read in with a set of 90 programs, one per dataset, with all of the values hardcoded. Generally, the format of these files had been consistent and entailed little maintenance from my part.

When I was approached by the survey team about making significant changes to the file structure, I realized that I didn't want to go through the process of searching thousands of lines of code just to find and update file structures nor to remove redundant values. To help solve this, I created a new process that involves a crosswalk, a few SAS macros, and a lot less lines of code.

This process has allowed my team to implement changes with very little work, saving us time and avoiding errors. As an added bonus, we now have a document that shows the structure of our files that is readable by a non-programmer.

In this paper, you will learn how I implemented this process, going from thousands of lines of SAS code to a crosswalk and a few short SAS macros, and see how to implement this with your own datasets.

READING TEXT FILES

An ASCII file, or a text file, contains information in human-readable format. In a typical SAS program, we use FILENAME, INFILE, and INPUT statements to read in the data. For the remainder of this paper we will assume that the input ASCII files are pipe-delimited.

SYNTAX EXAMPLE

```
filename read17 "H:\Practice\EXAMPLE_ASCII.A01";

data WORK.ADDR_TBL;
      attrib KEY      length=$14 format=$14. label="Key";
      attrib ADDR     length=$45 format=$45. label="Mailing Address";
      attrib ZIP      length=$5  format=$5.  label="Zip Code";
      attrib ADDRTYPE length=8   format=8.   label="Type of Address";

      infile read17 DELIMITER='|' DSD MISSOVER FIRSTOBS=2 LRECL = 32767;
      input KEY ADDR ZIP ADDRTYPE ;
run;
```

1) The FILENAME command provides a file reference to a given ASCII file.

   a. Note that in the example above the file extension of the ASCII file is *A??*, where *??* represents a number starting with 01. This is particular to the kind of file we are reading in; it is not essential to the process generally.

2) The INFILE statement references the file alias.

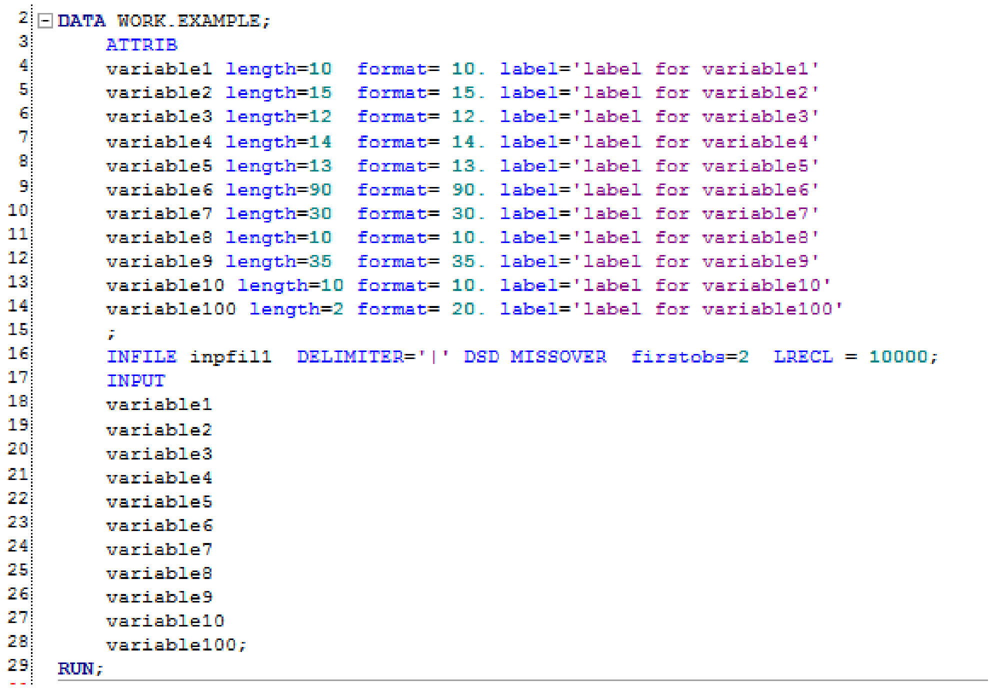
3) The ATTRIBUTE statement makes sure that SAS reads the file in a preferred format.

4) The INPUT makes sure that the columns are read in in the correct order.

MAINTAINING SEVERAL FILES – EFFICIENTLY

One of the main goals of this new approach is to reduce the number of lines of code in our core program. Having fewer lines of code means less code to maintain, and more importantly it helps you to better understand the overall flow of the program. Most lines of code in the original program are mostly attributed to two reasons:

1. ATTRIBUTE statements for each variable.
2. Listing the actual variables under the INPUT statement.

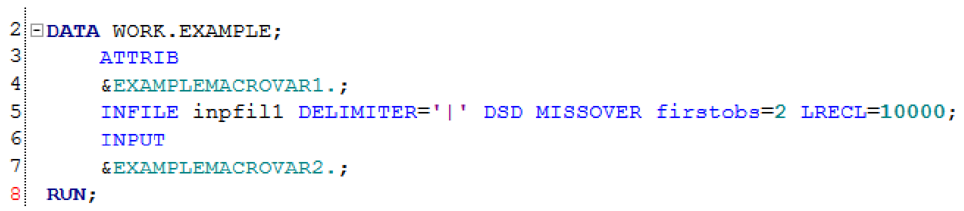


```
DATA WORK.EXAMPLE;
    ATTRIB
    variable1 length=10  format= 10. label='label for variable1'
    variable2 length=15  format= 15. label='label for variable2'
    variable3 length=12  format= 12. label='label for variable3'
    variable4 length=14  format= 14. label='label for variable4'
    variable5 length=13  format= 13. label='label for variable5'
    variable6 length=90  format= 90. label='label for variable6'
    variable7 length=30  format= 30. label='label for variable7'
    variable8 length=10  format= 10. label='label for variable8'
    variable9 length=35  format= 35. label='label for variable9'
    variable10 length=10 format= 10. label='label for variable10'
    variable100 length=2 format= 20. label='label for variable100'
    ;
    INFILE inpfil1  DELIMITER='|' DSD MISSOVER  firstobs=2  LRECL = 10000;
    INPUT
    variable1
    variable2
    variable3
    variable4
    variable5
    variable6
    variable7
    variable8
    variable9
    variable10
    variable100;
RUN;
```

Figure 1. Example of the original structure.

In Figure 1 we notice that the majority of the code are basic ATTRIBUTE statements and one lengthy INPUT listing. If we only had to maintain a couple of text files this would be a reasonable approach, however you can understand how tedious this might become if we had 90+ files with numerous columns.

This is where having a crosswalk can really benefit us. All those ATTRIBUTE statements and variables can easily be stored and managed in an external document. Once we have built the crosswalk, with the help of a handful of macros, we can achieve our goal of simplifying our core program.



```
DATA WORK.EXAMPLE;
     ATTRIB
     &EXAMPLEMACROVAR1.;
     INFILE inpfil1 DELIMITER='|' DSD MISSOVER firstobs=2 LRECL=10000;
     INPUT
     &EXAMPLEMACROVAR2.;
RUN;
```

Figure 2. Example of preferred structure.

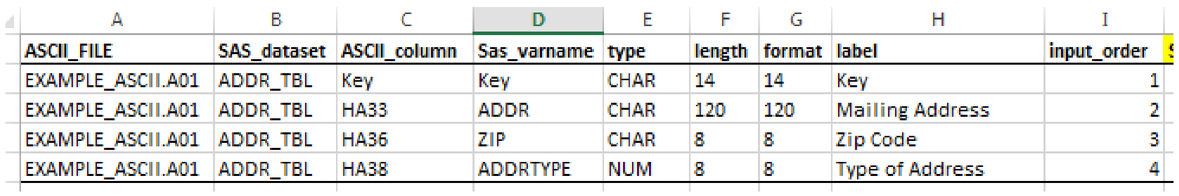
THE CROSSWALK

Some of the benefits of having a crosswalk include:

- Easily maintainable. You can easily modify files, variables, labels, formats, etc., while decreasing the chance of errors than stem from the alternative: searching lines of code in the program.
- Transparent. This format is more user-friendly and easily shared across teams.
- Self-documenting. The crosswalk is an easily understood list of variables and their attributes.

For our purposes the crosswalk is built in the form of a spreadsheet. It reads one row per variable per file and contains the following columns:

- ASCII_FILE and SAS_DATASET. This is the input file name and the SAS name for the output file respectively.
- ASCII_COLUMN and SAS_VARNAME. Often times the text file can have invalid SAS names or the column header is simply not descriptive enough, these columns will assist with these types of scenarios.
- TYPE, LENGTH, FORMAT, LABEL. Variable attributes.
- INPUT ORDER. Correct order of the column names. Having an input order column forces you to be attentive when you have to add or delete variable(s).



A	B	C	D	E	F	G	H	I
ASCII_FILE	SAS_dataset	ASCII_column	Sas_varname	type	length	format	label	input_order
EXAMPLE_ASCII.A01	ADDR_TBL	Key	Key	CHAR	14	14	Key	1
EXAMPLE_ASCII.A01	ADDR_TBL	HA33	ADDR	CHAR	120	120	Mailing Address	2
EXAMPLE_ASCII.A01	ADDR_TBL	HA36	ZIP	CHAR	8	8	Zip Code	3
EXAMPLE_ASCII.A01	ADDR_TBL	HA38	ADDRTYPE	NUM	8	8	Type of Address	4

Figure 3. Proposed Crosswalk.

Figure 3 shows our template. As described above, columns E through H contains the information needed for our ATTRIBUTE statement. Column D and Column I provides the necessary information for the INPUT statement.

For the remainder of this paper we will be referencing these columns names when talking about the crosswalk.

PUTTING IT ALL TOGETHER

Building the crosswalk is arguably the most time consuming part of the new process, however once the crosswalk is created, it only requires maintenance moving forward.

Once the crosswalk is complete, all we need is a handful of macros. More specifically, we need four utility macros to generate and store information about the input variables, attribute statements, and special missing assignments. These macros are quite straightforward and they only require a single parameter – the filename (which identifies the file). For more information about the syntax please refer to the appendix of this paper.

We will also include a main macro which will serve as a wrapper where we build the core syntax of the INPUT and INFILE statements.

In this section we assumed that the reader has some basic knowledge regarding SAS Macro execution and PROC SQL procedures. If you need more information about how this works, or alternative methods of accomplishing this, please see List Processing Basics (Fehd and Carpenter, 2007).

MAIN MACRO - READ_ASCII

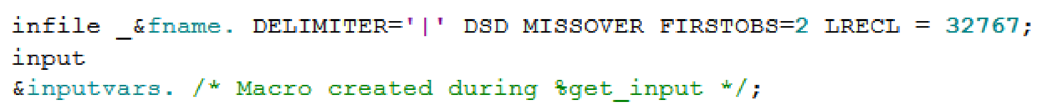
The main macro is called READ_ASCII, it executes the INFILE and INPUT statements using the information created by the utility macros. The current version of READ_ASCII requires four parameters:

1. **Fileloc**. Location of the ASCII file.
2. **Fname**. Name of the ASCII file to be read.
3. **Libout**. Target location to store the SAS dataset.
4. **Outsas**. Name of the SAS dataset.

```
%macro READ_ASCII(fileloc=,fname=,libout=,outsas=);
       *Step 1: Get Input Vars;
       %get_input(fname=&fname.)

       *Step 2: Get Attribute List;
       %get_attribs(fname=&fname.)

       *Step 3: Get variables for special missing;
       %get_specialmiss(fname=&fname.)

       *Step 4: Read in ASCII file using INFILE and INPUT;
       filename _&fname. "&fileloc.\EXAMPLE_ASCII.&fname.";

       data &libout..&outsas.;
       &attriblist. /* Macro created during %get_attrib */

       infile _&fname. DELIMITER='|' DSD MISSOVER FIRSTOBS=2 LRECL = 32767;
       input
       &inputvars. /* Macro created during %get_input */;

       *Step 5: Apply special missing if applicable;
       %if %length(&specmiss.) > 0 %then %do;
              %apply_specialmiss(fname=&fname.)
       %end;
       run;
%mend READ_ASCII;
```

BREAKDOWN

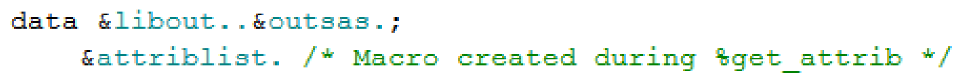
Step 1. Execute get_input macro. It creates a macro variable that is used during the INPUT statement. The crosswalk columns *SAS_VARNAME* and I*NPUT_ORDER* are referenced during this step. Figure 4 shows and example of the macro variable created: *inputvars*.



```
infile _&fname. DELIMITER='|' DSD MISSOVER FIRSTOBS=2 LRECL = 32767;
input
&inputvars. /* Macro created during %get_input */;
```

Figure 4. Example of INPUT statement.

Step 2. Execute get_attribs macro. It generates multiple ATTRIBUTE statements and it stores them in a macro variable. It utilizes *SAS_VARNAME*, *TYPE*, *LENGTH*, *FORMAT*, and *LABEL* from the crosswalk. Figure 5 shows the macro variable generated: *attriblist.*



```
data &libout..&outsas.;
    &attriblist. /* Macro created during %get_attrib */
```

Figure 5. Example of ATTRIBUTE statement.

Step 3. Execute get_specialmiss macro, which is custom project-specific code we need to execute during the input step. Its main purpose is to find the variables that need special missing assignments, it stores them in a macro variable. *SAS_VARNAME* and *SPECIALMISSING* from the crosswalk are used here. The example below is an example of how to create the macro variable called *specmiss*. This would be how any other custom code might be inserted for your own project.

```
%MACRO GET_SPECIALMISS(fname=);
       PROC SQL NOPRINT;
              SELECT DISTINCT SAS_VARNAME
              INTO :SPECMISS SEPARATED BY " "
              FROM WORK.CROSSWALK
              WHERE UPCASE(ASCII_FILE)="&fname."
              AND   UPCASE(SpecialMissing)='YES';
       QUIT;
%MEND GET_SPECIALMISS;
```

Step 4. Read the ASCII file using INFILE and INPUT statements. At this point in the process, we already have all we need to read in the file. We have the file location, file name, input variables and attributes created by the utility macros, and a target location and SAS name.

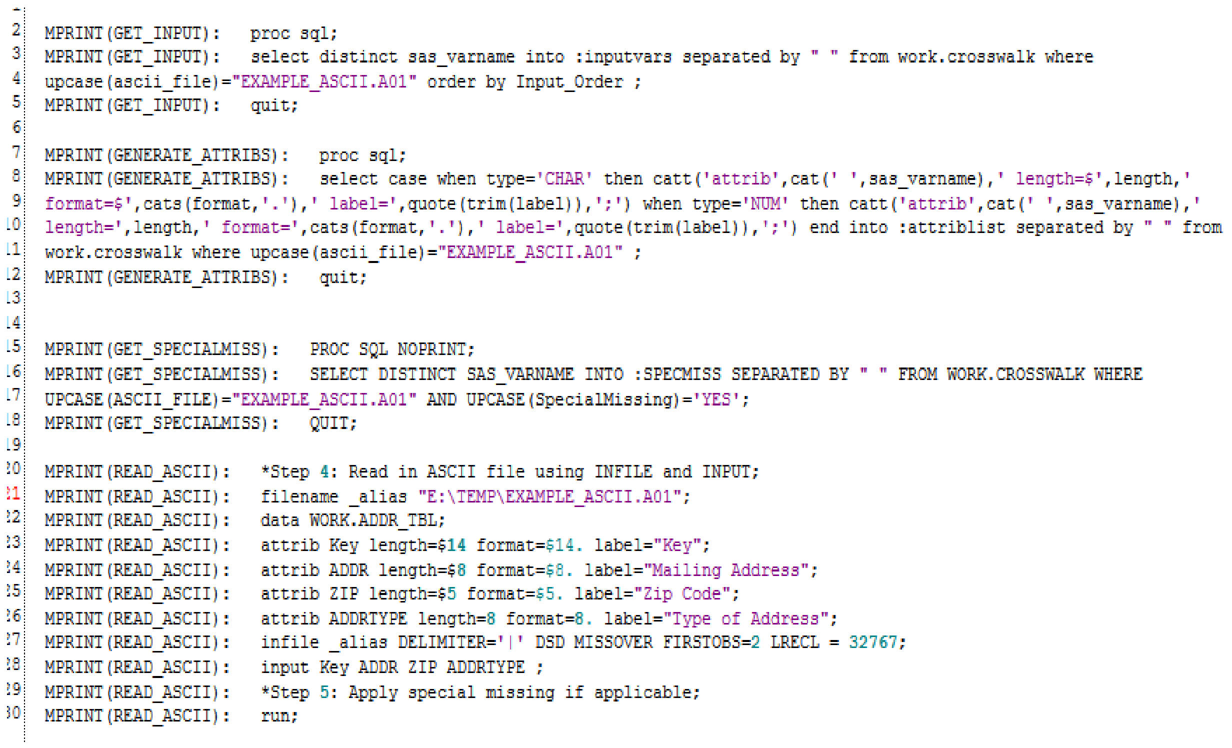
Step 5. This is where we apply the special missing values. We cannot assume that all the SAS files will require special missing assignments so we need to add a conditional macro statement. Again, this is custom for this particular project and could be removed or replaced with your own project's specific needs.

```
*Step 5: Apply special missing if applicable;
%if %length(&specmiss.) > 0 %then %do;
       %apply_specialmiss(fname=&fname.)
%end;
```

SAMPLE CALL AND LOG

Here's an example of how to execute READ_ASCII. The example below reads an ASCII file called *TEST_ASCII* and creates a SAS dataset called *ADDR_TBL* in the WORK library.

```
%READ_ASCII(fileloc=H:\Practice,fname=EXAMPLE_ASCII.A01,libout=WORK,outsas=ADDR_TBL)
```



```
MPRINT(GET_INPUT):   proc sql;
MPRINT(GET_INPUT):   select distinct sas_varname into :inputvars separated by " " from work.crosswalk where
upcase(ascii_file)="EXAMPLE_ASCII.A01" order by Input_Order ;
MPRINT(GET_INPUT):   quit;

MPRINT(GENERATE_ATTRIBS):   proc sql;
MPRINT(GENERATE_ATTRIBS):   select case when type='CHAR' then catt('attrib',cat(' ',sas_varname),' length=$',length,'
format=$',cats(format,'.'),' label=',quote(trim(label)),';') when type='NUM' then catt('attrib',cat(' ',sas_varname),'
length=',length,' format=',cats(format,'.'),' label=',quote(trim(label)),';') end into :attriblist separated by " " from
work.crosswalk where upcase(ascii_file)="EXAMPLE_ASCII.A01" ;
MPRINT(GENERATE_ATTRIBS):   quit;


MPRINT(GET_SPECIALMISS):   PROC SQL NOPRINT;
MPRINT(GET_SPECIALMISS):   SELECT DISTINCT SAS_VARNAME INTO :SPECMISS SEPARATED BY " " FROM WORK.CROSSWALK WHERE
UPCASE(ASCII_FILE)="EXAMPLE_ASCII.A01" AND UPCASE(SpecialMissing)='YES';
MPRINT(GET_SPECIALMISS):   QUIT;

MPRINT(READ_ASCII):   *Step 4: Read in ASCII file using INFILE and INPUT;
MPRINT(READ_ASCII):   filename _alias "E:\TEMP\EXAMPLE_ASCII.A01";
MPRINT(READ_ASCII):   data WORK.ADDR_TBL;
MPRINT(READ_ASCII):   attrib Key length=$14 format=$14. label="Key";
MPRINT(READ_ASCII):   attrib ADDR length=$8 format=$8. label="Mailing Address";
MPRINT(READ_ASCII):   attrib ZIP length=$5 format=$5. label="Zip Code";
MPRINT(READ_ASCII):   attrib ADDRTYPE length=8 format=8. label="Type of Address";
MPRINT(READ_ASCII):   infile _alias DELIMITER='|' DSD MISSOVER FIRSTOBS=2 LRECL = 32767;
MPRINT(READ_ASCII):   input Key ADDR ZIP ADDRTYPE ;
MPRINT(READ_ASCII):   *Step 5: Apply special missing if applicable;
MPRINT(READ_ASCII):   run;
```

Figure 6. Log for READ_ASCII

CONCLUSION

Reading and maintaining text files is a common task in a programmer's life but can easily become challenging and time consuming. Having a crosswalk allows you to become more efficient and accurate. In addition to the crosswalk, having a few simple macros can increase the flexibility of your program while greatly reducing the overall amount of code to maintain.

REFERENCES

Fehd, Ronald J. and Carpenter, Art. (2007). "List Processing Basics: Creating and Using Lists of Macro Variables". *Proceedings of the SAS Global 2007 Forum.* Cary, NC: SAS Institute, Inc. Available at: https://support.sas.com/resources/papers/proceedings/proceedings/forum2007/113-2007.pdf

APPENDIX

Below is an example of the core macros necessary to implement the ideas presented in this paper. The ASCII File and crosswalk can be found under the GitHub repository https://github.com/josercent.

```
%macro get_specialmiss(fname=);
       proc sql;
              select distinct sas_varname
              into :specmiss separated by " "
              from work.crosswalk
              where upcase(ASCII_FILE)=%upcase("&fname")
              and upcase(SpecialMissing)="YES";
       quit;
%mend get_specialmiss;

%macro generate_attribs(fname=);
       proc sql;
              select
```

```
                case
                when type='CHAR' then catt('attrib',cat(' ',sas_varname),'
length=$',length,' format=$',cats(format,'.'),' label=',quote(trim(label)),';')
                when type='NUM' then catt('attrib',cat(' ',sas_varname),'
length=',length,' format=',cats(format,'.'),' label=',quote(trim(label)),';')
                end
                into :attriblist separated by " "
                from work.crosswalk
                where upcase(ascii_file)=%upcase("&fname.");
        quit;
%mend generate_attribs;


%macro get_input(fname=);
        proc sql;
                select distinct sas_varname
                into :inputvars separated by " "
                from  work.crosswalk
                where upcase(ascii_file)=%upcase("&fname.")
                order by Input_Order /*Order is important here*/ ;
        quit;
%mend;

%macro READ_ASCII(fileloc=,fname=,libout=,outsas=);
%global attriblist inputvars specmiss;

        /* Step 1 Find and store input vars */
        %get_input(fname=&fname.);

        /* Step 2 Find and store attributes */

        %generate_attribs(fname=&fname.);

        /* Step 3 Find and store special missing values */
        %get_specialmiss(fname=&fname.);

        *Step 4: Read in ASCII file using INFILE and INPUT;
        filename _alias "&fileloc.\&fname.";
        data &libout..&outsas.;
                &attriblist. /* Macro created during %get_attribs */

                infile _alias DELIMITER='|' DSD MISSOVER FIRSTOBS=2 LRECL = 32767;
                input
                &inputvars. /* Macro created during %get_input */;

                *Step 5: Apply special missing if applicable;
                %if %length(&specmiss.) > 0 %then %do;
                        %apply_specialmiss(fname=&fname.)
                %end;
        run;
%mend READ_ASCII;

proc import file="E:\temp\ASCII_Crosswalk.xlsx"
out=crosswalk
dbms=xlsx
replace;
run;

options mprint;
%read_ascii(fileloc=E:\TEMP,libout=WORK,fname=EXAMPLE_ASCII.A01,outsas=ADDR_TBL)
```

ACKNOWLEDGEMENTS

I would like to thank Joe Matise for his invaluable assistance and for encouraging me to write this paper.

CONTACT INFORMATION

Jose Centeno

Programmer II

NORC at the University of Chicago

Centeno-jose@norc.org