Paper 344-2013

A Macro to Read in Medi-Span® Text Format Database by Data Dictionary

Sijian Zhang, VA Pittsburgh Healthcare System

ABSTRACT

Investigators often use commercial databases to obtain useful information for their researches. However, many companies do not offer the code for transferring the data files from their deliverable file format into the one used in customer's system. With many data files and variables, the data transfer process can be very tedious. If the databases vary in different versions, the transfer code revision can be another pain. This paper presents an approach to simplify the data transfer process of reading in Medi-Span drug information text data files by taking the advantage of macro programming and its data dictionary information. One of Medi-Span text data files, "MF2STR", is used as an example through this paper.

KEYWORD

Import text data files, data dictionary, macro, Medi-Span.

INTRODUCTION

Our center, Center for Health Equity Research and Promotion, VA Pittsburgh Healthcare System, has some research projects that involve medication identification data processing. Medi-Span drug information database was purchased for this purpose. Medi-Span is a leading provider of drug information for health care professionals worldwide.

To use Medi-Span database, the first thing is to transfer all the text data files into SAS® datasets. I contacted with the company's Technical Support to check if they had a SAS program for this purpose, and got its reply, "We do not have any programs for SAS. Our content is provided as raw text and the exercise of loading it is left to the consumer of the data to match their preferred schemas."

In Medi-Span database document, there is a variable information table for each text data file (see Figure 1). In this table, the information in the first four fields is useful for Data step to import the text data file. The first field, "Code", contains the field identifiers, or variable names. You may notice that the identifiers of this data file are all made of digits, which is not valid in SAS. So, we add a pre-fix "Medi_" to all identifiers to give new variable names in SAS datasets. The second field, "Data Element Name", actually contains the variable labels. The third field, "Record Position", gives the absolute positions for all fields. The fourth field, "Type/Length", contains the variable type (N – numeric, C – character) and its length. With the above information, all the text data files can be read in with Data step.

```
0000001076   D000000011300000650000000MG/30ML
0000007957   A00000011630000009500000MG
0000018545   D00000013160000004670000%
0000018594   A00000003680000000100000MG
0000023587   D00000030440000000600000MG
 ...
```

The first 5 rows in data file “MF2STR”

Code	Data Element Name	Record Position	Type/Length	Picture
1001	Ingredient Identifier	1-10	N/10	9(10)
1011	Reserve-1	11-12	C/2	X(2)
1013	Transaction CD	13-13	C/1	X
1014	Ingredient Drug ID	14-23	N/10	9(10)
1024	Ingredient Strength Value	24-36	N/13	9(8)V9(5)
1037	Ingredient Strength UOM (combined)	37-47	C/11	X(11)
1048	Ingredient Strength UOM (individual)	48-58	C/11	X(11)
1059	Volume Value	59-71	N/13	9(8)V9(5)
1072	Volume Unit of Measure	72-82	C/11	X(11)
1083	Reserve-2	83-112	C/30	X(30)

Variable information table for “MF2STR”

Figure 1: The first 5 rows in data file “MF2STR” and its variable information table

There are 30 text files in the package from Medi-Span, including 27 data files, 1 service statement file, 1 copyright file, and 1 dictionary file, “MF2DICT”, which contains variable attribute information of all the data files.

From the two approaches are presented in next section, you will see how the programming evolves from simple, but tedious to a little advanced, but highly efficient one.

APPROACH

1. *Data Step Only*

```
data medisas.MF2STR;
   infile "\\vhapthresearch1\MediSpan\...\MF2STR" truncover ①;
   input Medi_1001 1-10 Medi_1011 $11-12 Medi_1013 $13-13
         Medi_1014 14-23 Medi_1024 24-36 Medi_1037 & ② $37-47
         Medi_1048 & $48-58 Medi_1059 59-71 Medi_1072 & $72-82
         Medi_1083 & $83-112;

   label Medi_1001="Ingredient Indentifier"
         Medi_1011="Reserve-1"
         Medi_1013="Transaction Code"
         Medi_1014="Ingredient Drug ID"
         Medi_1024="Ingredient Strength Value"
         Medi_1037="Ingredient Strength UOM(combined)"
         Medi_1048="Ingredient Strength UOM(individual)"
```

```
            Medi_1059="Volume Value"
            Medi_1072="Volume Unit of Measure"
            Medi_1083="Reserve-2";
run;

  .
  .
  .
```

The INFILE option “truncover” ① will prevent from reading in wrong data from next record if there are missing values at the end of the current record. Since there are single blacks in some character values, the modifier “&” ② used in INPUT statement is to treat single blacks (delimiters by default) as valid contents of a character value and the double or more blanks together as a delimiter.

This approach works. But you may not be very enthusiastic to go down this way when you realize that you have to repeat such a Data step 27 times, find all the arguments form variable information tables or data dictionary file, and key them in INPUT and LABEL statements. How about if those argument values will be changed in the next version of the database? In situation like this, Macro should be the right way to go. The next approach will show you how Macro works.

2. Macro Application

In order to streamline the data transfer process, two components should be included: 1) macro to cut off the repeated coding effort; and 2) data dictionary information to be used to get variable names, field positions and variable labels for the arguments of INPUT and LABEL statements automatically.

Before looking into the macro, let’s have a view at the data dictionary file “MF2DICT”, see Figure 2. Four of the fields are used in the macro: Field Identifier (C001) for variable name, Field Description (C005) for variable label, Field Type (C040) for variable type, and Field Length (C041) for value position calculation. “NewName” is a created variable by adding “Medi_” to all field Identifiers during MF2DICT.sas7bdat generation in a Data step like in Approach I. The values of “NewName” will be the variable names in resulting SAS datasets.

	Field Identifier (C001)	Field Description (C005)	Field Type (C040)	Field Length (C041)	NewName (NewName)
1	1001	Ingredient Indentifier	N	10	Medi_1001
2	1011	Reserve-1	C	2	Medi_1011
3	1013	Transaction Code	C	1	Medi_1013
4	1014	Ingredient Drug ID	N	10	Medi_1014
5	1024	Ingredient Strength Value	N	13	Medi_1024

Figure 2: Screenshot of the first 5 observations of SAS dataset “MF2DICT”

You can see that the variable information in Figure 1 and 2 is similar. But the table in Figure 1 is a PDF file, and Figure 2 is a SAS dataset; and “MF2DICT” has only the field length, not field position as in Figure1.

The following is the macro to transfer text data files into SAS datasets. Macro variable “datasetName” passes the input data filename and gives the name of resulting SAS dataset; “initial” passes the first letter of the field identifiers. For example, since all field identifiers in“MF2STR” start with “1”, the SAS dataset MF2STR.sas7bdat is generated after running the macro %txt2sas(MF2STR,1). Since all field identifiers in one data file start with the same initial character or digit, which is different from other data files, the field identifier initial is used to select the related variable information from the dictionary file. The information of filenames and initials can be found in the database document and dictionary file.

```
%macro txt2sas(datasetName,initial);
/* Use data dictionary to set up arguments for INPUT and LABEL */
data datavar;
   set medisas.mf2dict;

   * Select variables by initial *;
   if substr(C001,1,1)="&initial";          ❶

   * Set up Start and End positions of variables *;
   retain Start 1;
   Start+lag(C041);                         ❷
   End=Start+C041-1;                        ❸

   * Set up arguments for INPUT statement *;
   if C040='C' & end-start>=3
        then Var_Pos=NewName||' & $'||strip(Start)||'-'||strip(end);
   else if C040='C' & end-start<3
        then Var_Pos=NewName||' $'||strip(Start)||'-'||strip(end);
   else Var_Pos=NewName||' '||strip(Start)||'-'||strip(end);          ❹

   * Set up auguments for LABEL statement *;
   Label=NewName||'='||'"'||strip(C005)||'"';                         ❺
run;

/* Assign the INPUT arguments to macro variable Input_Vars */
proc transpose data=datavar out=var_pos(drop=_name_);                 ❻
   id C001;
   var var_pos;
run;

data input_vars;
   set var_pos;
   length Comb_Vars $2000;
```

```
   array vars _character_;
   do i=1 to dim(vars);
      Comb_Vars=strip(Comb_Vars)||' '||strip(vars[i]);
   end;
   call symput("Input_Vars", Comb_Vars);          ❼
run;

/* Assign the LABEL arguments to macro variable Labels */
proc transpose data=datavar out=var_label(drop=_name_);
   id C001;
   var Label;
run;

data labels;
   set var_label;
   length Comb_Labels $2000;
   array vars _character_;
   do i=1 to dim(vars);
      Comb_Labels=strip(Comb_Labels)||' '||strip(vars[i]);
   end;
   call symput("Labels", Comb_Labels);                  ❽
run;

/* Transfer text data file to SAS dataset */
data medisas.&datasetName;
   infile "\\vhapthresearch1\MediSpan\...\&datasetName"  truncover;
   input &Input_Vars;                                   ❾
   label &Labels;
run;
%mend txt2sas;

%txt2sas(MF2STR,1);
  .
  .
  .
```

Let's run %txt2sas(MF2STR,1) step by step to see how this macro works. In the first Data step of the macro, the dictionary file "MF2DICT.sas7bdat" is read in, the field identifiers and related information are selected for certain data file according to their initials ❶, and the four new variables are generated: Start – field start position ❷, End – field end position ❸, Var_Pos – concatenation of name, modifier, type, and position for each variable ❹, and Label – concatenation of name, equal sign, and label text for each variable ❺, see Figure 3.

	Field Identifier (C001)	Field Description (C005)	Field Type (C040)	Field Length (C041)	NewName (NewName)	Start (Start)	End (End)	Var_Pos (Var_Pos)	Label (Label)
1	1001	Ingredient Indentifier	N	10	Medi_1001	1	10	Medi_1001 1-10	Medi_1001="Ingredient Indentifier"
2	1011	Reserve-1	C	2	Medi_1011	11	12	Medi_1011 $11-12	Medi_1011="Reserve-1"
3	1013	Transaction Code	C	1	Medi_1013	13	13	Medi_1013 $13-13	Medi_1013="Transaction Code"
4	1014	Ingredient Drug ID	N	10	Medi_1014	14	23	Medi_1014 14-23	Medi_1014="Ingredient Drug ID"
5	1024	Ingredient Strength Value	N	13	Medi_1024	24	36	Medi_1024 24-36	Medi_1024="Ingredient Strength Value"
6	1037	Ingredient Strength UOM(combined)	C	11	Medi_1037	37	47	Medi_1037 & $37-47	Medi_1037="Ingredient Strength UOM(combined)"
7	1048	Ingredient Strength UOM(individual)	C	11	Medi_1048	48	58	Medi_1048 & $48-58	Medi_1048="Ingredient Strength UOM(individual)"
8	1059	Volume Value	N	13	Medi_1059	59	71	Medi_1059 59-71	Medi_1059="Volume Value"
9	1072	Volume Unit of Measure	C	11	Medi_1072	72	82	Medi_1072 & $72-82	Medi_1072="Volume Unit of Measure"
10	1083	Reserve-2	C	30	Medi_1083	83	112	Medi_1083 & $83-112	Medi_1083="Reserve-2"

Figure 3: Screenshot of temporary dataset “datavar” generated after running %txt2sas(MF2STR,1)

The purpose of this step is to generate the two variables: “Var_Pos” and “Label”, which contain the information to be used in INPUT and LABEL statements in the last Data step of the macro. Let’s look at how “Var_Pos” values are manipulated to form the actual arguments for INPUT; the same process is applied to “Label” values. In order to put all values of “Var_Pos” in one line, “Var_Pos” column is transposed into a one observation dataset ❻, see Figure 4.

	_1001	_1011	_1013	_1014	_1024	_1037	_1048	_1059	_1072	_1083
1	Medi_1001 1-10	Medi_1011 & $11-12	Medi_1013 $13-13	Medi_1014 14-23	Medi_1024 24-36	Medi_1037 & $37-47	Medi_1048 & $48-58	Medi_1059 59-71	Medi_1072 & $72-82	Medi_1083 & $83-112

Figure 4: Screenshot of temporary dataset “var_pos” in running %txt2sas(MF2STR,1)

Then, all variables in this dataset are concatenated and assigned to variable “Comb_Vars”. In the last statement, the following concatenated value is assigned to a macro variable “Input_Vars” ❼:

> Medi_1001 1-10 Medi_1011 $11-12 Medi_1013 $13-13 Medi_1014 14-23 Medi_1024 24-36 Medi_1037 & $37-47 Medi_1048 & $48-58 Medi_1059 59-71 Medi_1072 & $72-82 Medi_1083 & $83-112

After the same process, the following string is assigned to a macro variable “Labels” ❽:

> Medi_1001="Ingredient Indentifier" Medi_1011="Reserve-1" Medi_1013="Transaction Code" Medi_1014="Ingredient Drug ID" Medi_1024="Ingredient Strength Value" Medi_1037="Ingredient Strength UOM(combined)" Medi_1048="Ingredient Strength UOM(individual)" Medi_1059="Volume Value" Medi_1072="Volume Unit of Measure" Medi_1083="Reserve-2"

With the two macro variables, we come to the last Data step. INFILE reads in the text data file “MF2STR”; the above two macro variables fill the arguments of INPUT and LABEL statements ❾; in the end, a permanent dataset, “MF2STR.7bdat”, is generated, see Figure 5.

	Ingredient Indentifier (Medi_1001)	Reserve-1 (Medi_1011)	Transaction Code (Medi_1013)	Ingredient Drug ID (Medi_1014)	Ingredient Strength Value (Medi_1024)	Ingredient Strength UOM(combined) (Medi_1037)	Ingredient Strength UOM(individual) (Medi_1048)	Volume Value (Medi_1059)	Volume Unit of Measure (Medi_1072)	Reserve-2 (Medi_1083)
1	1076		D	113	65000000	MG/30ML		.		
2	7957		A	1163	9500000	MG		.		
3	18545		D	1316	4670000	%		.		
4	18594		A	368	100000	MG		.		
5	23587		D	3044	600000	MG		.		

Figure 5: Screenshot of the first 5 observations of the permanent dataset MF2STR.sas7bdat

The data transfer task will be done after running this macro to all the 27 text data files. In the future, if new variables are added, or some old variables are dropped, or field type / length are changed, which are not uncommon in different versions of databases, this program will still work fine without any revision since the data dictionary file is used to set up the required parameters automatically.

CONCLUSION

Obviously, the macro approach is the recommended solution. In the programming process, the data dictionary information plays a key role in setting up variable names, calculating the field positions and assigning the variable labels. The macro approach cuts off the repeated coding effort significantly, in the meantime, save the code modifying time if there are some variable changes in different database versions since the data dictionary information is directly applied in the process. Not only Medi-Span data users will find the macro approach helpful, others can also apply this idea to similar data transfer tasks. Databases are different, however, the way of using data dictionary information in a data transfer macro is always worth trying. It will increase your coding efficiency and make the program maintenance easy.

REFERENCES

1. Txt2sasg Delimited Text Files into SAS® 9, http://support.sas.com/techsup/technote/ts673.pdf
2. Medi-Span website: http://www.medispan.com/index.aspx

ACKNOWLEDGMENT

I would like to thank Dr. Xinhua Zhao for her valuable inputs and kind support while writing this paper.

CONTACT INFORMATION

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

Sijian Zhang, MD, MS, MBA
Research Health Science Specialist
Biostatistics and Informatics Core
Center for Health Equity Research and Promotion
VA Pittsburgh Healthcare System
7180 Highland Drive (151C-H)
Pittsburgh, PA 15206
sijian.zhang@va.gov