SESUG 2022 Paper 149

PROC IMPORT: Controlling SAS® Data Types and Character Variable Lengths When Reading CSV Files

Imelda C. Go, PhD, Cognia, Inc. and
Abbas S. Tavakoli, DrPH, MPH, ME, University of South Carolina

ABSTRACT

The task at hand is we have a SAS data set and the client needs a CSV file containing all the data in that SAS data set. Validate the CSV file to make sure that it contains exactly what is in the SAS data set. Using PROC IMPORT to read CSV files is convenient. However, relying solely on PROC IMPORT features does not always produce data sets with the desired properties, which can make combining two or more data sets difficult. This paper goes through an example of how you can control or adjust variable types in a data set generated by PROC IMPORT by using GUESSINGROWS=NO, reading as the first record a character string that defines the length of variables, and then reading the CSV file. The technique can be useful when we do not have the source SAS data sets and are given CSV files, which we have to read and then combine into a single SAS data set.

INTRODUCTION

Since we have the data set from which the CSV originated from, we have the information required to build a custom DATA STEP to read the contents of a CSV file. We can also look at the log and see the PROC IMPORT code that SAS uses to read the data. We can reuse and modify this data to suit our needs. Still another option is the example presented in this paper.

There are good reasons to want to control the variable attributes of PROC IMPORT outcomes.

1. To combine two or more SAS data sets with a SET statement in a DATA Step, SAS requires that variables from different SAS data sets have the same name and type.
2. For character variables, we also need to worry about truncation if the same character variables from different SAS data sets have different lengths.
3. Only numeric variable types can be used with procedures that require numeric variables. If a numeric variable only has blank values, that will be defined as character.

For a CSV file, we can use a GUESSINGROWS value to help SAS determine the variable type. However, its current maximum value is 32,767, which may or may not be sufficient for your needs.

1. If SAS reads data as a character, all the data will be stored as a character value.
2. If SAS sees only blank data, the data will be read as a character type, which may or may not be what we want.
3. If SAS sees only numbers, the variable will be numeric. However, we may still want the variable to be character instead of numeric.
4. We may experience data loss when SAS decides to read data as numeric when there are character values. Character values will appear as missing values.
5. We may experience truncation. Once the length has been defined, longer character values will be truncated up to the defined length.

PROGRAMMING STEPS FOR EXAMPLE

1. The SAS data set is called DataSource.
2. Export its contents into a CSV file using PROC EXPORT.
3. Create a comma-delimited string consisting of C's or N's that is based on the length and type of each DataSource variable and save it as a file to be used as input later. This will be called &file1 and will be used to set the lengths of the character variables.
4. Export DataSource contents into a CSV file using PROC EXPORT. This will be called &file2.
5. Define a composite FILENAME statement with two parts:

   ```
   FILENAME DATAFILE (&file1, &file2);
   ```

6. Use PROC IMPORT to read the composite DATAFILE CSV file with GETNAMES=NO statement. This will force variable names to be var1, var2, …. Because the first record has only letters for values, the variable types of all variables will be character. From the data set produced by PROC IMPORT, we can discard the 1st record, which has already served its purpose to set the character lengths. We can isolate the observed header (2nd record) and the observed data (all records following the 2nd record).
7. To validate the header, we change the var1, var2, var3, … names to original names by using the values provided in the PROC CONTENTS of DataSource. We transpose the header data so that it can be compared with the PROC CONTENTS data set from DataSource. We can use PROC COMPARE to compare the headers.
8. To validate the data, we convert all the character variables into numeric ones based on attributes of DataSource variables as specified in PROC CONTENTS data set. We also need to rename the variables to the original names in DataSource. We can use PROC COMPARE to compare the data.

proc import file=" C:\Users\imelda.go\Desktop\filelayout.xlsx" out=EXPCSVHeader (keep="variable name"n "data element number"n where=("data element number"n ne .) rename=("variable name"n=COL1)) dbms=xlsx replace;	Get the data with the expected layout labels. These are the values that should appear in the header row for the CSV file.
proc contents data=DataSource noprint out=EXPDataLabels (keep=name type length varnum) varnum ; run; proc sort data=EXPDataLabels; by varnum; proc sql noprint; select count(*) into :n from EXPDataLabels; quit; %let numvar=%cmpres(&n);	Get the data with the expected data labels from the SAS data source. Put the number of records for expected data labels into macro variable numvar.
data lengthstrings; length var1-var&numvar. $10000; retain var1-var&numvar. ; set EXPDataLabels end=eof; array varlist (*) $ var1-var&numvar.; if type=2 then varlist{varnum}=repeat('C',length); else if type=1 then varlist{varnum}=repeat('N',8); if eof then output; keep var:; drop varnum; run; proc export data=lengthstrings dbms=csv file="&sandbox.\lengthstrings.txt" replace; putnames=no;	Create a single string that controls the length of the character variables for the CSV file. Using the PROC CONTENTS data, we know what is the maximum length for each character variable in the SAS data source. We will create a string of C's, which repeat till the length of each character variable. For numbers, there will just be an N. This data is written out with no header (PUTNAMES=NO). (To keep this example simple, we will assume the numbers just need a length of 8.)
data renamestring; length renamestring $32767; retain renamestring ''; set EXPDataLabels end=eof; renamestring=strip(renamestring)||" """||strip(name)||"""n=var"||strip(varnum); if eof then call symput("renamestring",renamestring); run;	This step read the expected data labels and creates a string that will be used to rename the sequence of variables.
data NumericVars; set EXPDataLabels (where=(type= 1)); run; data convertstring; length convertstring $30000; retain convertstring ''; set NumericVars end=eof; convertstring=strip(convertstring)||"%nrstr(%convert)"||"(var"||strip(put(varnum,8.))||");"; if eof then call symput("convertstring",convertstring); run; ***count # of conversions; proc sql noprint; select count(*) into: nconvert from convertstring; quit; %put &nconvert;	Determine which variables from source data set are numeric. Using these data, create a string that will be used to convert the numeric variables into strings. Put the number of numeric variables in marco variable nconvert.

filename datafile ("&sandbox.\CSVdata._lengthstrings.txt","&filename") ;	Use a filename, to be used with PROC IMPORT, that consists of two parts. The first part is the string that controls the length of the character variable. The second part is the CSV data file itself.
proc import out=CSVdata file=datafile dbms=csv replace; getnames=no; run;	Read the data specified in this 2-part filename but DO NOT get the variable names from the header (GETNAMES=NO). This will result in a data file that has var1, var2, var3, … as variable names. Because the 1st row only contains letters, automatically all variables will be a character type. The first row was created such that the character variables will not be truncated.
proc contents data=CSVdata noprint out=OBSCSVdataContents (keep=name type length varnum) varnum ; **run**;	Create a SAS data set with the observed CSV data contents.
%macro convert(var); if strip(&var)=:'' then &var.num=.; else &var.num=input(&var,8.); drop &var; rename &var.num=&var; **%mend** convert;	This macro generates code that converts the character values from the CSV data into numeric ones.
data EXPCSVdata; set datasource (drop=&dropvarlist); rename &renamestring ; **run**;	Rename the variables in the CSVdata so that it will have the same expected variable names. Use the &renamestring.
data OBSCSVHeader OBSCSVdata; set CSVdata; if _n_=1 then do; delete; end; else if _n_=2 then output OBSCSVHeader; else output OBSCSVdata; **run**;	Because of the 2-part file name we used, we know that the 1st record was to control the length of the character variables. The 2nd record is the header and records after that contain the data.
proc transpose data=OBSCSVheader out=OBSCSVheader ; var var:; **run**; **proc compare** data=EXPCSVHeader compare=OBSCSVheader listall outall out=diffheader; title COLOR=RED "DataSource HEADER ROW"; **run**;	Transpose the CSV header so that it goes from 1 record to several. The observed CSV header can now be compared with the Expected values specified by the file layout.
options errors=5; %if &nconvert=0 %then %do; **data** ObservedData; set OBSCSVdata; **run**; %end; %else %do; **data** ObservedData; set OBSCSVdata; &convertstring; **run**; %end;	It is possible that the source data set has no numeric variables and that there are no variables in the observed CSV data that needs to be converted. Use the convertstring to the necessary character variables into numeric ones.

`` proc datasets lib=work memtype=data noExpectedDatant; modify ExpectedData ; attrib _all_ label=''; attrib _all_ format=; attrib _all_ informat=; modify ObservedData ; attrib _all_ format=; attrib _all_ informat=; run; proc sort data=ExpectedData; proc sort data=ObservedData; run; proc compare data=ExpectedData compare=ObservedData; run; %MEND CSVValidation; ```	Data sets often attributes which could be flagged as a difference when using PROC COMPARE. To minimize these differences, we can remove the labels, formats, and informats that might trigger differences even when we are only interested in the data values. We run a final PROC COMPARE to make sure that the contents of the data source are the same as the CSV data.

CONCLUSION

The SAS programming language offers different tools to help us with our work. By understanding how procedures work, we can more easily design automated solutions that help us achieve the desired result.

CONTACT INFORMATION

Imelda C. Go
Cognia, Inc.
imelda.go@cognia.org

Abbas S. Tavakoli
University of South Carolina
abbas.tavakoli@sc.edu

TRADEMARK NOTICE