Paper SAS1778-2018

A Macro for Ensuring Data Integrity When Converting SAS® Data Sets

Richard D. Langston, SAS Institute Inc.

ABSTRACT

This paper describes the %COPY_TO_NEW_ENCODING macro, which ensures that a SAS® data set does not experience data loss when being copied to a new encoding. The focus is on the FORMAT procedure's CNTLOUT data sets and the problems that can occur when converting from single-byte encoding like Windows Latin1 to multi-byte encoding like UTF-8.

INTRODUCTION

If you need to convert the character variables in a SAS® data set from one encoding to another, it is possible that the current length setting for one or more character variables might be insufficient to hold the converted values. This macro will verify the maximum length needed for each character variable so that the new data set will have the sufficient lengths to avoid any truncation.

UNDERSTANDING THE TRANSCODING PROBLEM

Consider as an example a SAS data set created using the Windows Latin1 encoding. You have a character variable with alphabetic characters that are not a-z or A-Z, such as the French acute accent e character (é) in the name Beyoncé. The text takes 7 bytes, which are in hexadecimal 4265796F6E63E9, with the é being hexadecimal E9 in Windows Latin1 encoding. But if you transcode this text to UTF-8 encoding, it requires 8 bytes, 4265796F6E63C3A9, because é is represented by 2 bytes, C3A9, in UTF-8.

If you simply have

```
data orig; x='Beyoncé'; run;
data new(encoding=utf8); set orig; run;
```

you will encounter errors due to the truncation and data loss:

```
ERROR: Some character data was lost during transcoding in the dataset WORK.NEW. Either the data
       contains characters that are not representable in the new encoding or truncation occurred
       during transcoding.
```

You will need to know a length to choose in order to accommodate the possible expansion of characters. Had you known that 8 bytes were needed, you could have overridden the length this way, providing the LENGTH statement before the SET statement, and then you would avoid the error.

```
data orig; x='Beyoncé'; run;
data new(encoding=utf8); length x $8; set orig; run;
```

THE %COPY_TO_NEW_ENCODING MACRO

The % COPY_TO_NEW_ENCODING macro will determine the minimum length needed for each character variable so that all data values can expand successfully. The macro does this by traversing through the entire data set, and performing a transcoding on each character value via the KCVT function. If the macro determines that at least one character variable needs to have a revised length, then it generates a series of LENGTH statements for all the variables.

We begin with the macro definition. We will have 3 arguments: the "from" data set name, the "to" data set name, and the encoding to use for the "to" data set:

```
%macro copy_to_new_encoding(from_dsname,to_dsname,new_encoding);
```

There will be new variables created within DATA steps where the input variables can be arbitrary, and we do not want to collide with input variable names. Therefore, we introduce a prefix to be prepended to our new variable names to avoid collisions. The prefix value should have some set of nonsense characters so that there will be no name collisions.

```
* Use a prefix to ensure unique variable names;
%let prefix=goobly;
```

The file containing the LENGTH statements will be unconditionally %INCLUDEd, so we want to make sure the file exists (as a temporary file) and that it has at least one blank line in it.

```
* create a LENGTH statement file with just a blank line;
filename lngtstmt temp;
data _null_; file lngtstmt; put ' '; run;
```

We will be unconditionally deleting the data set temp2, although it might otherwise not be created, so we create a dummy version here so that it does exist.

```
* create a temp2 that might be replaced before being deleted;
data temp2; x=1; run;
```

Now we get the names, lengths, types, and positions of all variables in the input data set. The CONTENTS procedure can do this for us, and with NOPRINT no listing is produced. Note that the WHERE= option is used so that only character variables (type=2) are written to our output data set, because we are only concerned with character variables. Note that although we don't subsequently need the type variable, we must have it in the KEEP= list because it is used in the WHERE clause. Note that the output data set will be sorted by name by PROC CONTENTS, which is the sort order we need.

```
* Get the names, lengths, types, and positions of all variables;
proc contents data=&from_dsname
     out=temp1(keep=name type length npos where=(type=2)) noprint; run;
```

At this point, the data set temp1 will exist. It will have zero observations if there are no character variables, and it will have at least one observation if there are any character variables. We can set the macro variable &nchars from the NOBS= option, which will tell us the observation count, which is the same as the number of character variables. Note that we don't actually need to execute the SET statement, because the NOBS= variable is set during DATA step compilation time. Note also the use of the SYMPUTX function, which allows for the second argument to be a numeric variable without complaint, unlike SYMPUT.

```
* macro variable nchars indicates the number of character variables found;

%global nchars;
data _null_;
     call symputx('nchars',nchars);
     stop;
     set temp1 nobs=nchars;
     run;
```

At this point, the &nchars macro variable will either be zero (meaning no character variables) or nonzero (meaning there are character variables). There is no need to execute most of the code if there are no character variables, and no revision of lengths will be needed.

```
* Revision is only possibly necessary if there are character variables;
```

```
%if &nchars %then %do;
* macro variable revise will be set to 1 if a revision is needed;
%global revise;
%let revise=0;
```

We now read through the input data set. We use a _TEMPORARY_ array to hold the lengths of all the character variables, and we use an old-style array with the special _CHARACTER_ keyword so that we can traverse all the character variables. For _N_=1 (initializing as of the first observation) we obtain the declared lengths of the character variables via the VLENGTH function. Note that each length is initially stored as its negative, so that only those values that change will be set to positive.

Each character value is transcoded via KCVT. The LENGTHC function determines the returning length of the transcoded trimmed value. We don't use LENGTH because that relies on session-encoded blanks, which might not be used in the transcoded value. If that returned length exceeds the currently known maximum length for the variable, then the known maximum length is revised.

And then once all values for all observations are examined, we write out the temp2 data set containing the revised lengths and the variable names. And the macro variable &revise is set to 1 if any lengths were revised.

```
data temp2(keep=&prefix._name &prefix._length
           rename=(&prefix._name=NAME &prefix._length=LENGTH));
     set &from_dsname end=&prefix._eof;
     retain &prefix._revise 0;
     array &prefix._charlens{&nchars} _temporary_;
     array &prefix._charvars _character_;

     * get the lengths of all the character variables, set as negative ;
     if _n_=1 then do over &prefix._charvars;
        &prefix._charlens{_i_}= -vlength(&prefix._charvars);
        end;

     * transcode all values and see if the lengths increase;
     do over &prefix._charvars;
        &prefix._l = lengthc(
           kcvt(trim(&prefix._charvars),"&new_encoding.")
           );
        if &prefix._l > abs(&prefix._charlens{_i_}) then do;
           &prefix._charlens{_i_} = &prefix._l;
           &prefix._revise = 1;
           end;
        end;

     * output any varnames and revised lengths;
     if &prefix._eof and &prefix._revise;
     call symputx('revise',1);
     length &prefix._name $32 &prefix._length 8;
     do over &prefix._charvars;
        if &prefix._charlens{_i_} > 0 then do;
           &prefix._name = vname(&prefix._charvars);
           &prefix._length = &prefix._charlens{_i_};
           output temp2;
           end;
        end;
     run;
```

At this point, if the &revise macro variable is 0, this means that although there were character variables in the input data set, none of them needed their lengths revised. If &revise is 1, then at least one character variable needs a revised length, and temp2 will contain the names and revised lengths.

```
* if any lengths revised we will create a LENGTH statement;
%if &revise %then %do;
```

The temp data set is updated with the revised lengths via a MERGE by name as the key.

```
* merge in the revised lengths;
proc sort data=temp2; by name; run;
data temp1; merge temp1 temp2; by name; run;
```

The generated LENGTH statements should be in the same order as the original input data set, meaning npos order, so we sort by npos to get the original order.

```
* sort back to npos to maintain the original order;
proc sort; by npos; run;
```

Now we can generate the LENGTH statements. There is one for each variable, using the original length or revised length as appropriate. We must ensure that a $ appears for character variables, and that the NLITERAL function should be used in case the variable might need it in the expression. For example, the variable 'a b'n (a blank b) must be expressed as an n-literal in the LENGTH statement syntax. Note that NLITERAL will not produce an n-literal if it is not necessary.

```
* generate a LENGTH statement for all variables in order;

data _null_; set temp1; file lngtstmt mod;
     len = cats(ifc(type=2,'$',' '),length);
     stmt = catx(' ','length',nliteral(name),len,';');
     put stmt;
     run;
%end;
%end;
```

At this point, we have a temporary file lngtstmt, which contains either just a blank line (no length revision needed) or a series of LENGTH statements, so the file can be %INCLUDEd in our DATA step to create the new data set with the new encoding.

```
* create the new data set with the original or revised lengths;
data &to_dsname(encoding=&new_encoding);
     %include lngtstmt/source2;
     set &from_dsname;
     run;
```

To be good macro citizens, we should clean up. So we clear the lngtstmt file because it is no longer needed, and we delete the temporary data sets. Note that the DELETE procedure allows us to list multiple data sets.

```
* cleanup;

filename lngtstmt clear;
proc delete data=temp1 temp2; run;

%mend copy_to_new_encoding;
```

EXAMPLE OF USE

```
/* encoding should be wlatin1 */
proc format cntlout=new;
     value mytest 1='Beyoncé';
     run;

/* should fail */
data try1(encoding=utf8); set new; run;

/* should succeed, with label length being increased */
%copy_to_new_encoding(new,try2,utf8);

data _null_; set try2; l=vlength(label); put l=; run;
```

The log shows the following:

```
NOTE: Data file WORK.TRY2.DATA is in a format that is native to another
host, or the file encoding does not match the session encoding. Cross
Environment Data Access will be used, which might require additional CPU
resources and might reduce performance.

l=8
```

The original length of the label variable is 7, but the % COPY_TO_NEW_ENCODING macro changes the length to 8 to accommodate the expansion of the special character in UTF8.

It should be noted that format catalogs do not have an encoding indicator, as SAS data sets do. It is assumed that the format catalog contains data using the same encoding as the current session encoding.

COMPARISON WITH THE CVP ENGINE

It should be pointed out that another way to create a copy of the data set into a new encoding and to accommodate increased lengths is the CVP engine. However, the CVP engine uses a fixed ratio of increased lengths without examining the data, therefore causing all character variables to become arbitrarily larger.

To use our example, but with CVP instead, running with a WLATIN1 encoding:

```
138        proc format cntlout=new;
139             value mytest 1='Beyoncé';

NOTE: Format MYTEST has been output.
140             run;

NOTE: The data set WORK.NEW has 1 observations and 21 variables.

141  libname revised cvp "%sysfunc(getoption(work))" outencoding=utf8;

WARNING: Libref REVISED refers to the same physical library as WORK.

142        data _null_; set new;
```

```
143             l=vlength(start); put l=;
144             l=vlength(end); put l=;
145             l=vlength(label); put l=;
146             run;

l=16
l=16
l=7

147         data _null_; set revised.new;
148             l=vlength(start); put l= ;
149             l=vlength(end); put l=;
150             l=vlength(label); put l= ;
151             run;

l=24
l=24
l=11
```

We see that the declared lengths of start and end have increased (from 16 to 24) although they didn't need to, and label has a larger length (11 versus 8) than the one chosen by the macro.

CONCLUSION

If your data contains characters that will need a larger byte count representation in another encoding, then you can use the % COPY_TO_NEW_ENCODING macro so that the minimum increase for the character variables is used.

CONTACT INFORMATION

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

Rick Langston
SAS Instititute Inc.
Rick.Langston@sas.com