PAPER TS05

KEPT IN TRANSLATION: AVOIDING DATA LOSS AND OTHER PROBLEMS WHEN CONVERTING JAPANESE DATA

Steve Prust, Covance, Leeds, UK

ABSTRACT

This paper details a method of translating and converting data from a non-western character set into English. The example used is where the original data was Japanese text in SAS datasets (the output also to be in SAS datasets). The paper will look at aspects of the technical environment, analyses how best a translation might be done, and presents the detailed approach of the translation and conversion. Quality control issues and SAS techniques are also discussed.

INTRODUCTION

Asked to translate data from a different character set is challenging but even more difficult is when you cannot even open the dataset in the first place! However, getting from this unpromising starting point to translated datasets was not a difficult as might have been feared.

TECHNICAL ENVIRONMENT

SAS has the ability, via National Language Support, to store data in many different character sets. For character sets with a large number of characters (e.g. Japanese and Chinese) two bytes are used to represent character (whereas ASCII needs only one byte to represent a character), hence the terminology of Double-Byte Character Set (or DBCS). SAS version 9 supports in DBCS the UTF-8 scheme (UTF=Unicode Transformation Format).

The SAS datasets that appear in this example were encoded using the 'Shift JIS', a character encoding for the Japanese language.

When installing SAS 9.1 there is the option to install a DBCS and Unicode version – when installed this appears on the start menu as “SAS 9.1 (English with DBCS and Unicode support)”. This version of SAS is able to process the S-JIS (for example) encoded datasets without needing any further set up. If using Viewtable – for exanple – the Japanese characters appear properly on screen. [note that if processing DBCS text it is necessary to use the double-byte equivalent of functions such as LENGTH [the double byte function equivalent of LENGTH is KLENGTH – for more details on these functions see in “Double-byte Character String Functions” SAS Help].

Note: There is an occasional problem that seems to affect DBCS SAS users in terms of the display font in SAS. This renders the DBCS text unreadable and appears to be related to font installation issues. Some installations seen by the author have had this problem and there are some reports from other users too. There is presumably a correct method to resolve this but as it concerns only the onscreen display it does not affect the techniques described in this paper. The only place where it becomes a problem is at the checking stage but a useful workaround for Viewtable has been posted on one of the SAS bulletin boards (SAS-L posting by Randy Herbison, 1 April 2008, subject “Re: SAS viewer window: set font permanently”, see http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0804a&L=sas-l&P=9142).

ANALYSIS

In a SAS DBCS dataset there are three types of text that may need converting:

- data values
- variable labels
- format values

Text may be either phrases, words or single characters (for instance if the script in question has an analagous construct to “initials”).

As may be expected on any database the individual text strings may be repeated many times. It is not sensible to merely print out a database and require each value to be translated as this would multiply the effort required several-

fold and introduce a greater possibility of error. Eliminating from the translation requirement the duplication of phrases makes the task shorter and more robust. This implies the identification of unique strings.

Text strings may be repeated in several different places and across the three types identified above. However, to remove the context from a text string could mean it was harder to provide an accurate translation. For this reason the translations have retained as much context as possible. This meant for data values analysing and presenting, for each variable separately, the unique values to be translated. Thus the translator would have a sent of text strings that were likely to have a common theme (the variable name and variable label might also provide some useful context). For formats it was possible to list the variables (and labels) where the formats were used. For variable labels the unique text strings were presented across the entire set of datasets.

Translation of the text into English needed to be in machine readable form. Because of the advantages of word processors over spreadsheets in terms of word processing, recovery, undo facility etc. the worksheets were created as RTF documents via ODS (and so both Word and Open Office compatible).

The drawback of creating RTF documents was that this meant reading the data back into SAS would not be as straightforward. Two alternatives were either a) to copy and paste each translation worksheet table into an Excel workbook, or b) to devise a means of communication between SAS and Word using DDE and/or a Word macro. The first option was chosen for simplicity.

APPROACH

The stages in the approach were as follows:

- determine which data values, variable labels, and format values needed translation
- tag the unique text strings for translation
- create translation worksheets
- pass the worksheets for translation
- read the translated text into SAS
- apply the translations

DETERMINING WHAT TEXT NEEDED TRANSLATION

As descibed in the analysis above the three types of text are handled slightly differently in terms of finding unique text strings for translation.

Variable labels were obtained from PROC CONTENTS. The procedure created and output dataset. The dataset was sorted by the label text. Each unique label value was analysed for whether it contained only western characters. Such labels were removed from the translation process.

A similar process involving PROC FORMAT was used for format text. PROC FORMAT has the ability to create “format control” datasets that describe a format fully. Again the resultant dataset was analysed for text containing only western characters. Any format containing only western characters was removed from the trasnlation process (formats that were part-western and part-DBCS were kept in their entirety)

All the unique data values from each dataset were evaluated (a looping macro was used to make this processing easier - some sample code for this is shown in Appendix 1). Any variable that had some non-western text had all its values retained for translation.

TAGGING THE UNIQUE VALUES FOR TRANSLATION

In order to be capable of loading the translated text back onto the datasets it was necessary to have a means of uniquely identifying each text string. The identification was done using the following:

- for formats: the the format name and start / end values
- for variable labels: the dataset name and the variable name
- for data values: an identification number

Whereas for format and variable label text strings the unique identifiers already existed it was necessary to create an identification number for data value text strings and add that to the datasets. For each variable requiring translation a new numeric variable was added called <variable name>_ID containing the unique identifier. The following example illustrates the technique:

If a dataset contains the following values :

	VOL	ACTION	CONT	DAYS
1	101	中止	良好	22
2	102	その他	良好	12
3	106	中止	良好	15
4	107	中止	不良	18
5	109	その他	良好	21
6	115	継続	良好	17

The Action and Cont variables require translation. Action has three unique values (その他, 中止 and 継続) Cont has two (良好 and 不良). Two identifier variables are added to the dataset like so:

	VOL	ACTION	CONT	DAYS	ACTION_ID	CONT_ID
1	101	中止	良好	22	2	1
2	102	その他	良好	12	1	1
3	106	中止	良好	15	2	1
4	107	中止	不良	18	2	2
5	109	その他	良好	21	1	1
6	115	継続	良好	17	3	1

Finally the original variables were removed and new datasets were created using the dataset option of encoding=any (meaning they could be read using “normal” SAS). Using the above example again would have produced this:

	VOL	DAYS	ACTION_ID	CONT_ID
1	101	22	2	1
2	102	12	1	1
3	106	15	2	1
4	107	18	2	2
5	109	21	1	1
6	115	17	3	1

CREATING TRANSLATION WORKSHEETS

Using the text identified for translation, together with the unique identifiers, the translation worksheets were created using ODS RTF. For example, for variable labels:

```
options orientation=landscape;
filename _temp_ "<project output location>\formats.rtf";
ods noresults;
ods listing close;
ods rtf file=_temp_  ;
```

```
title  'Translation worksheet - Formats - Project Ref: xxxx';
proc print data=format_con noobs label;
  var fname   / style(data)={cellwidth=30%};
  var id      / style(data)={cellwidth=8%};
  var fval    / style(data)={cellwidth=30% font_face="MS Mincho"};
  var blank   / style(data)={cellwidth=30%};
  label fname = 'Format name';
  label id    = 'Id value(s)';
  label fval  = 'Formatted value';
  label blank = 'Translation';
run;
ods rtf close;
ods results;
ods listing;
filename _temp_;
```

Resulting in output such as :

Translation worksheet - Formats - Project Ref: xxxx

Format name	Id value(s)	Formatted Value	Translation
GENDER	1	男	
GENDER	2	女	
YESNO	1	有	
YESNO	2	無	

These worksheets were then passed for translation.

READING THE TRANSLATED WORKSHEETS INTO SAS.

On receipt of the translated documents the table of translations was copied into an Excel worksheet and saved.

The reading of the Excel worksheets was done using PROC IMPORT using code such as:

```
proc import out=ae_trans
            datafile= "<location>\ae_trans.xls"
            dbms=excel2000 replace;
     getnames=yes;
run;
```

Previous versions of SAS and Excel have not always been trouble-free when doing this type of import. Mindful of the possibilities of errors, especially data truncation and character/numeric conversion, checks were made for both these things as well as for unreadable characters (in case some Japanese text had somehow been placed into the translation column.

The check for data truncation was to find the length of the variable that SAS had assigned in PROC IMPORT and print out all data values with a length at or near this variable length. This list of values could then be manually checked against the translators worksheets for truncation.

```
proc contents data=<dataset imported from excel> noprint nodetails out=fmtn_d;
run;
data _null_;
  set fmtn_d;
  if name = 'TRANSLATION' then
    call symput('tran_len',compress(put(length,best.)));
run;
```

```
data _null_;
  set <dataset imported from excel>;
  tranlen = length(translation);
  if tranlen + 5 > &tran_len then do;
    output;
    put "Length of item near max, check for truncation: " tranlen= translation=;
  end;
run;
```

All translations were text strings, thus making the check for conversion to a numeric variable easy (via PROC CONTENTS)

Unreadable characters were checked for by searching for non-alphanumeric etc. characters and reporting any exceptions.

The only check that found any errors was that for unreadable characters. These were all resolvable fairly easily. Moof these errors were a result of western characters being represented in a Japanese font (the Japanese font MS Mincho has character representation of the western character set e.g.

ａｂｃｄｅｆｇｈｉｊｋｌｍｎｏｐｑｒｓｔｕｖｗｘｙｚ１２３４５６７８９０）

APPLYING THE TRANSLATIONS.

Once the Excel worksheets were read and validated the translations could be applied to the formats, labels and data values.

FORMATS

Formats were created by creating a format control dataset. For example:

```
data control;
  set <dataset imported from excel>;
  length fmtname $8 start end $12 label $200;
  fmtname = format_name;
  start   = id_value_s_;
  end     = start;
  type    = 'C';
  label   = translation;
run;
proc format library=<libname> cntlin=control;
run;
```

DATA LABELS

Data labels were applied using PROC DATASETS together with a MODIFY statement. For example:

```
proc datasets library=<library> nodetails nolist;
  modify <dataset>;
    label <variable> = "new label";
run;quit;
```

DATA VALUES

Data values were applied by taking each dataset and successively merging on all the translated variables by the <varname>_ID values. The code was written such that any partial merge matches would be reported for investigation using code such as :

```
data <new dataset>;
  merge <interim ds> (in=_base) <translations> (in=_trans);
  by <var>_id;
  rename translation = var;
  if _base and not _trans then do;
    if not missing(<var>_id) then
      put "WARNING: unexpected non-match for variable <var>_id, value="<var>_id;
  end;
  if not _base and _trans then
    put "WARNING: unexpected extra-match for variable <var>_id, value="<var>_id;
```

```
run;
```

Partial matches should not of course occur. But, given how the nature of the translation worksheets (i.e. that data other than the translation column could have been accidentally amended) it was sensible to make this check. As it happened there were no partial matches.

FINAL STEP

The final stage of the conversion was twofold: remove the <var>_ID values and put the dataset variables back in the original order.

The code used for this is shown in Appendix 2.

At the conclusion of this step the datasets were fully converted into SAS

REVIEW

The sheer number of words and variables (over 12,000 words and about 80 different variables) made the task onerous. Risk analysis evaluated the status of the work at a level of medium. Given this it was pleasing to complete the work with few if any problems. The method has proved to be robust, effective and efficient.

The systematic approach to the translation scope and processes was key to the success of the project. Getting the technical environment correctly set-up allowed the individual elements of the task to be tackled in succession. Each of the elements had their complexities but these were not at the highest level of difficulty.

There is potential to improve the method of loading the translations (rather than move text from Word to Excel and then to SAS). This element of the process was cumbersome and has some higher levels of risk.

CONTACT INFORMATION

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

Steve Prust

Covance

Springfield House, Hyde Street

Leeds, UK LS2

Email:stephen.prust@covance.com

APPENDIX 1. LOOPING CODE

The following code illustrates the use of PROC CONTENTS and the Data step to create macro variables for iteratively processing datasets and for each variable to be translated within the current dataset.

```
/* find _id variables */
proc contents data=<library>._all_ nodetails noprint out=jcon;
run;

/* find  id variables */
data idvars;
  set jcon;
  length id $32;
  if length(name) < 3 then
    delete;
  if upcase( substr(name, length(name)-2 ,3) ) = '_ID' then do;
    id = name;
    name = substr(name, 1, length(name)-3 );
    output;
  end;
  keep memname name;
run;

proc sort data=idvars;
  by memname name;
run;

data _null_;
  set idvars end=last;
  by memname;
  retain dsid 0 varid 0;
  if first.memname then do;
    dsid = dsid + 1;
    varid = 0;
    call symput('cdsnam' || compress(put(dsid,best.)) , trim(memname) );
  end;
  varid = varid + 1;
  call symput('cdsn' || compress(put(dsid,best.)) || 'v' ||
compress(put(varid,best.)) , trim(name) );
  call symput('cdsn' || compress(put(dsid,best.)) || 'i' ||
compress(put(varid,best.)) , trim(id) );
  if last.memname then
    call symput ('cdsit' || compress(put(dsid,best.)), compress(put(varid,best.)));
  if last then
    call symput('cdsnum',compress(put(dsid,best.)));
run;

/* apply id values */
%macro apply_id;
  %do i = 1 %to &cdsnum;     /* process each dataset */
    %put processing ds &i name &&cdsnam&i;
    %do j = 1 %to &&cdsit&i;      /* process each variable in dataset */
       %put importing id values for item &j name &&cdsn&i.v&j by &&cdsn&i.i&j;

        /* code to do import and apply translations goes in here */

    %end;
  %end;
  run;quit;
%mend;
%apply_id;
```

APPENDIX 2. CODE TO RE-APPLY VARIABLE ORDER AND SORT ORDER

The following code is used to re-apply the same order of variables and the same sort order as the original datasets to the converted datasets.

```
/* find what order variables were on original datasets */
proc contents data=<original library>._all_ nodetails noprint out=jocon;
run;

/* create a list of variables suitable for use in PROC SQL */
proc sort data=jocon;
  by memname varnum;
run;
data jocon2;
  set jocon;
  by memname;
  length list $1000;
  keep memname list;
  retain list;
  if first.memname then
    list = name;
  else
    list = trim(list) || ', ' || name;
  if last.memname then output;
run;

/* create a list of sort variables suitable for use in PROC SQL */
proc sort data=jocon out=jocon3;
  by memname sortedby;
  where sortedby;
run;
data jocon4;
  set jocon3;
  by memname;
  length sortlist $1000;
  keep memname sortlist;
  retain sortlist;
  if first.memname then
    sortlist = name;
  else
    sortlist = trim(sortlist) || ', ' || name;
  if last.memname then output;
run;

/* create macro variables for each dataset: ds name, variable list, sortlist */
data jocon5;
  merge jocon2 jocon4;
  by memname;
run;
data  null ;
  set jocon5 end=last;
  call symput('fds'||compress(put(_n_,best.)),trim(memname));
  call symput('fvar'||compress(put(_n_,best.)),trim(list));
  call symput('fsort'||compress(put(_n_,best.)),trim(sortlist));
  if last then
    call symput('fn',compress(put(_n_,best.)));
run;

/* copy over datasets with same variable order as originally */
%macro create;
  %do i = 1 %to &fn;
    %put processing &i &&fds&i;
    proc sql;
      create table <target library>.&&fds&i as select
        &&fvar&i
```

```
      from <intermediate library>.&&fds&i
      %if &&fsort&i ^= %then %do;
        %str(order by &&fsort&i)
      %end;
      ;
    quit;
  %end;
%mend;
%create;
```