PhUSE 2012

Paper CC03

Dynamically generating macro invocations using SAS keyboard abbreviations

Tom Van Campen, SGS Life Science Services, Mechelen, Belgium
Benny Haemhouts, SGS Life Science Services, Mechelen, Belgium

ABSTRACT

A keyboard abbreviation is a text string recognized by the SAS® Enhanced Editor which can be replaced by a text that you pre-define.

This could be particularly useful for storing pieces of code you often need but are tedious to memorize.
A practical example: a macro and its parameters can be inserted in the SAS Enhanced Editor simply by entering the macro name.

These abbreviations can be shared by means of exporting them to a .kmf-file.

The standard method for entering and updating keyboard abbreviations is a manual, possibly labor intensive task with a risk for errors.

Generating the .kmf-file via an automated process could nullify these drawbacks.
This implies that you have to decipher the .kmf-file algorithm.

This paper will show you how to generate such a file.

INTRODUCTION

Many papers already exist on the use, manual creation and benefits of the SAS® keyboard abbreviations.
The number and changeability of these abbreviations might lead to an unmanageable manual process.

This paper will focus on the advantage to have the keyboard abbreviations generated automatically.
The concept will be illustrated by using a clearly defined example, generating/managing SAS macro invocations.
This could then also be expanded to include for example macro descriptions.

This paper was created using SAS 9.2 in a Windows environment.

PROBLEM DESCRIPTION

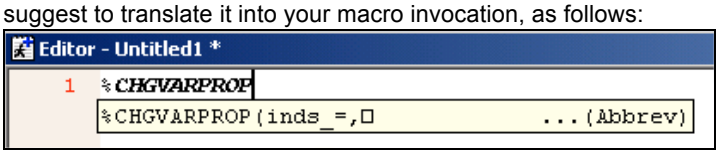
Given the large amount of validated SAS macros within a company, the amount of macro parameters is too extensive to know by heart.
Therefore a process which helps in the retrieval of these parameters could be both useful and time-saving.
For instance a macro which changes the properties of a variable on your dataset: CHGVARPROP.
This macro has following parameters: inds_, var_, newvar_, varlabel_, varfmt_ and varinfmt_.

PREFERRED SOLUTION

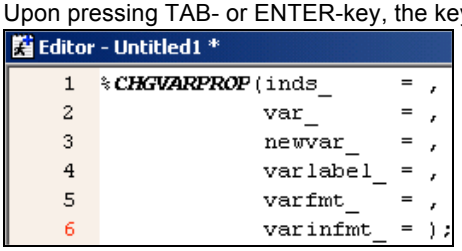
Keyboard abbreviations offer the most user friendly approach and are embedded in the SAS Enhanced Editor.

AN EXAMPLE

When the text %CHGVARPROP is pre-defined as a keyboard abbreviation, upon entry, SAS Enhanced Editor will suggest to translate it into your macro invocation, as follows:

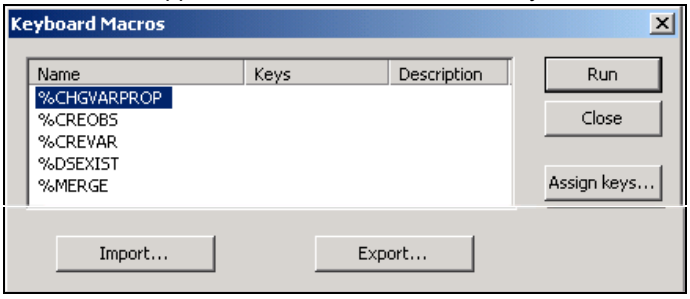


Upon pressing TAB- or ENTER-key, the keyboard abbreviation will be translated into a pre-defined replacement text:



SHARING KEYBOARD ABBREVIATIONS BETWEEN USERS/MACHINES

An advantage of keyboard abbreviations is that they can be exported and imported via .kmf-files.
The standard approach is via the menu tools – keyboard macros – macros:



Note: when exporting you need to select all macros you wish to export.
Note: importing is done one .kmf-file at a time.

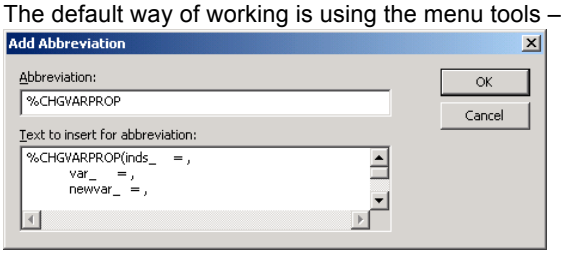
TECHNICAL LIMITATIONS

Keyboard abbreviations are case sensitive. In the example above *%chgVarProp* will not be recognized.
There is a maximum of 401 keyboard abbreviations (with a sequence number ranging from 0 to 400).
The replacement text is limited to 30.000 characters.

THE STANDARD APPROACH

CREATION OF KEYBOARD ABBREVIATIONS

The default way of working is using the menu tools – Add Abbreviation:



Note: text alignment looks bad, this is due to the font type (which differs from the SAS Enhanced Editor).

DISADVANTAGES

Even though this can be achieved via copy/pasting, it will still be a manual repetitive process.

Given a large amount of macros, this will result in a labor intensive task both in creation and maintenance.
It will also introduce a risk for errors.

A MORE DYNAMIC APPROACH

EXTRACTING MACRO DEFINITION

Via an automated process, macro definitions are extracted from the .sas files residing in our SAS macro production environment. This is possible by detecting the macro definition in the file, e.g.:

```
%macro CHGVARPROP(inds_=, var_=, newvar_=, varlabel_=, varfmt_=, varinfmt_=);
```

These definitions are then translated into macro invocations with all parameters and stored in a standardized SAS dataset. An example:

	_KMF_seq	_KMF_name	_KMF_inStr
1	4	%CHGVARPROP	%CHGVARPROP(inds_ = , var_ = , newvar_ = , varlabel_ = , varfmt_ = , varinfmt_ =);

Note: each line in the _KMF_inStr variable (except the last line) ends with following combination: byte(13) byte(10).
The SAS byte function converts an integer into a specific ASCII character:

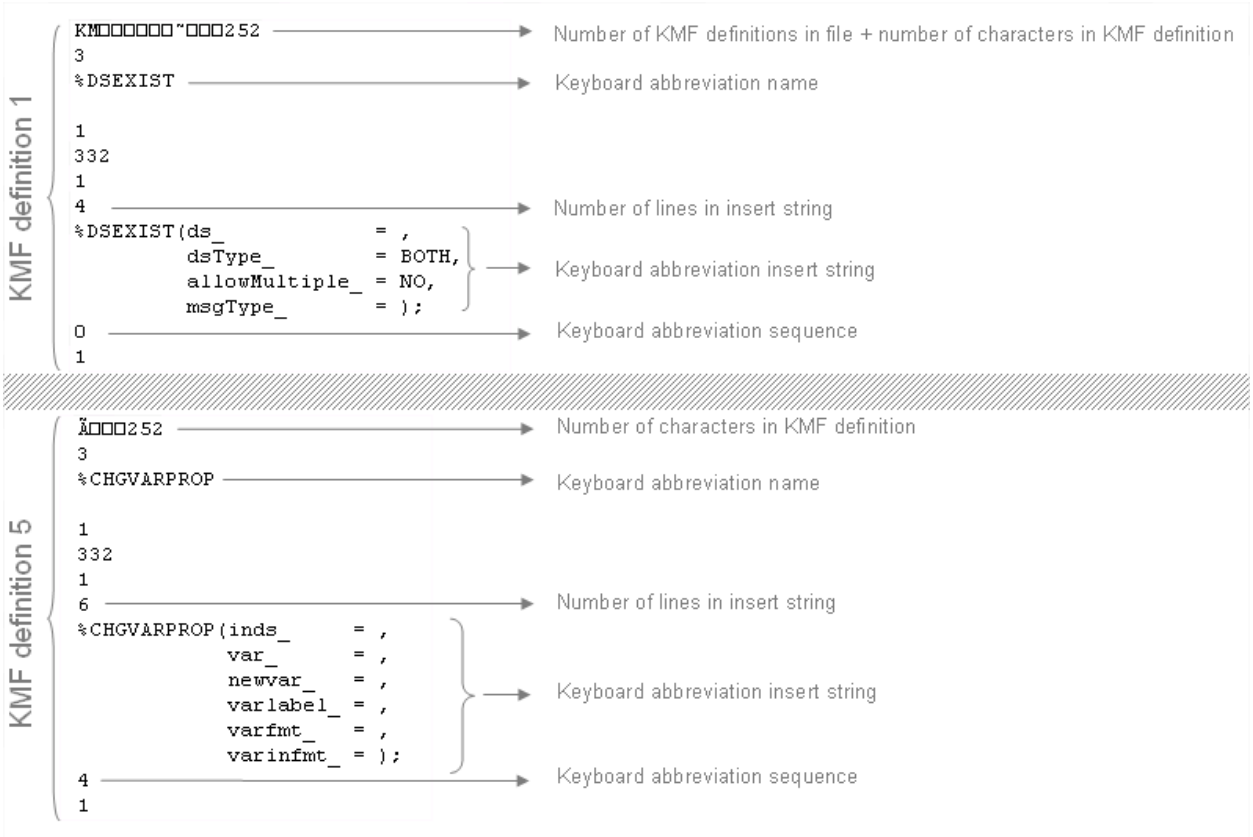
- *byte(13) translates into* carriage return.
- *byte(10) translates into new line feed.*
- *note that this integer can range from 0 to 255 following the ASCII collating sequence.*

Note: _KMF_seq should start from zero.

This file can now be fed into the .kmf-file generator.

.KMF-FILE EXAMPLE

To be able to create a .kmf-file you need to understand its algorithm.
An example .kmf-file:



Note: for the first KMF definition, extra information is shown on the number of KMF definitions stored in the file.

Note: sections which are not annotated are static for keyboard abbreviations.

.KMF-FILE ALGORITHM

Number of KMF definitions

The number of KMF definitions is preceded by the fixed text "KM□ □ " or "KM" byte(0) byte(2).
The actual number of KMF definitions is translated from a regular number into a four byte text. (e.g. Ó□ □ □):

- fourth byte: number of KMF definitions divided by 256^3 (rounded down).
- third byte: remainder of division by 256^3, divided by 256^2 (rounded down).
- second byte: remainder of division by 256^2, divided by 256 (rounded down).
- first byte: remainder of division by 256.

Note: third and fourth bytes are always zero since maximum 401 KMF definitions are supported by SAS.
Examples translating regular number into coded form:

- 211 KMF definitions: "KM" byte(0) byte(2) **byte(211) byte(0)** byte(0) byte(0) -> "KM□ □ Ó□ □ □ "
- 256 KMF definitions: "KM" byte(0) byte(2) **byte(0) byte(1)** byte(0) byte(0) -> "KM□ □ □ □ □ □ "
- 291 KMF definitions: "KM" byte(0) byte(2) **byte(35) byte(1)** byte(0) byte(0) -> "KM□ □ #□ □ □ "

Number of characters

The number of characters is followed by the fixed text "252".
The number of characters is translated using the same method as the number of KMF definitions.
Note: third and fourth bytes are always zero since maximum length will be around 30.000 characters, due to the restriction on insert string.
Example translating regular number into coded form:

- 500 characters: **byte(244) byte(1)** byte(0) byte(0) "252" -> "ô□ □ □ 252"

Number of characters count

Number of characters is:

- number of characters in the keyboard abbreviation name (e.g. %CHGVARPROP has 11 characters)
- number of characters in the insert string, note that each line ends with a byte(13) and byte(10)
- both the sequence and line counter can vary in length (e.g. sequence 20 has a two character length)
- for the static parts, 21 characters will always have to be counted

Note: also a description can be provided via the .kmf-file, if so these characters also have to be counted. This description is located in the .kmf-file below the keyboard abbreviation name.

Keyboard abbreviation name

The source text to be replaced.

Number of lines in insert string

The number of lines in the insert string.

Keyboard abbreviations insert string

The replacement text.

Keyboard abbreviation sequence

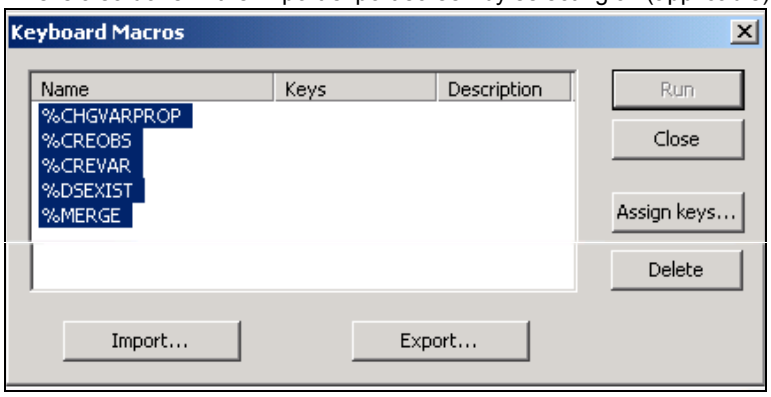
The sequence in which the abbreviation are loaded.

.KMF-FILE GENERATOR (SAS MACRO)

The macro will import the standardized SAS dataset, apply the .kmf-file algorithm and create a .kmf-file on a central location accessible to all SAS users.
The SAS macro code can be found in Appendix 1.

PREPARING THE IMPORT OF THE .KMF-FILE

Since the import of the .kmf-file will ask confirmation for each overwrite of a keyboard abbreviation, we recommend removing all previous versions before importing the updated .kmf-file.
This is also done in the import/export screen by selecting all (applicable) macros and pressing the *Delete button*.



IMPORTING THE GENERATED .KMF-FILE

Importing the generated .kmf-file in SAS, is also done via the import/export screen, using the *Import button*.
After import the user has all up-to-date abbreviations available for use.

CONCLUSION

Keyboard abbreviations are very useful and don't necessarily imply a lot of work when generated automatically.

ACKNOWLEDGEMENTS

We would like to thank the Statistics management and team for the support during the creation of this paper.

RECOMMENDED READING

- Byte() and rank() function in SAS.
- ASCII table (Google™).
- Papers on keyboard abbreviations (Google™).

CONTACT INFORMATION

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

Tom Van Campen, tom.vancampen@sgs.com, Biostatistical Coordinator / Senior Biostatistical Analyst
Benny Haemhouts, benny.haemhouts@sgs.com, Senior Biostatistical Analyst
SGS Life Science Services, www.sgs.com/en/Life-Sciences

APPENDIX 1: .KMF-FILE GENERATOR (SAS MACRO CODE)

```
%*************************************************************************************************;
%* Owner         : SGS Life Science Services                                                     *;
%* Macro name    : KMFgenerator                                                                  *;
%* Macro version : Light version for PhUSE paper                                                 *;
%* Developer     : Tom Van Campen, Biostatistical Coordinator                                    *;
%* Creation date : 18JUN2012                                                                     *;
%*                                                                                               *;
%* Description :                                                                                 *;
%*    This macro creates .kmf-files based upon information from a dataset with fixed structure *;
%*                                                                                               *;
%* Explanation of parameters :                                                                   *;
%*    inDS     = input SAS dataset containing information to be translated in .kmf-files       *;
%*    outLoc_  = location where the .kmf-files will be created                                 *;
%*    outFile_ = name of the .kmf-file                                                         *;
%*    del_     = delete temporary views                                                        *;
%*                                                                                               *;
%* Change control :                                                                              *;
%*    Initials Date       Description                                                           *;
%*    -------- ---------- ------------------------------------------------------------------- *;
%*                                                                                               *;
%*************************************************************************************************;

%macro KMFgenerator(inDS_=, outLoc_=, outFile_=, del_= YES);
   %* activate options compress if this is not yet the case (and store current setting) ;
   %let  KMG compress = %sysfunc(getoption(compress, keyword));
   options compress = YES;

   %let _KMG_abort = N;

   %* STAGE 0: ERROR HANDLING ;
   %* ---------------------- ;
   %* error handling and parameter checking is not foreseen (not in the scope of this paper) ;

   %* STAGE 1: PREPARE ALL INFORMATION TO CONSTRUCT LINES FOR KMF FILE ;
   %* ------------------------------------------------------------------ ;
   %if &_KMG_abort = N %then %do;
      %* determine number of definitions provided ;
      %let _KMG_dsid     = %sysfunc(open(&inDS_,in));
      %let _KMG_noDef    = %sysfunc(attrn(&_KMG_dsid,nobs));

      %* retrieve length of name and instr variable provided for use further on ;
      %let _KMG_varno    = %sysfunc(varnum(&_KMG_dsid, _KMF_name));
      %let _KMG_nameLen  = %sysfunc(varlength(&_KMG_dsid, &_KMG_varno));
      %let _KMG_varno    = %sysfunc(varnum(&_KMG_dsid, _KMF_instr));
      %let _KMG_instrLen = %sysfunc(varlength(&_KMG_dsid, &_KMG_varno));
      %let _KMG_rc       = %sysfunc(close(&_KMG_dsid));

      %* define flexible parts in the KMF file ;
      %* details on calculations below are in the paper ;
      data _KMG_prep / view = _KMG_prep;
         retain seqno _KMF_name _KMF_instr noLinesIS;
         set &inDS_;

         %* only for first record, define number of definitions ;
         if _N_ = 1 then do;
            %* translate regular numbers into bytes ;
            noDef     = &_KMG_noDef;
            tmpByteD2 = floor(noDef/256);
            tmpByteD1 = noDef - (tmpByteD2*256); *TVC: use mod function ;
            noDefByte = byte(tmpByteD1) || byte(tmpByteD2) || byte(0) || byte(0);
         end;

         %* define number of lines in the insert string ;
         noLinesIS = strip(put(count(_KMF_inStr, byte(13)) + 1, best32.));

         %* put sequence number into char ;
```

```
      seqno = strip(put(_KMF_seq, best3.));

      %* define the number of characters in definition ;
      %* - length of name and insert string changes ;
      %* - length of sequence number and number of lines in insert string also changes ;
      %* - 21 is the amount of characters in the definition that does not change ;
      noChar     = sum(length(strip( KMF name)), length(strip(_KMF_inStr)),
                       length(noLinesIS), length(seqno), 21);
      tmpByteC2  = floor(noChar/256);
      tmpByteC1  = noChar - (tmpByteC2*256);
      noCharByte = byte(tmpByteC1) || byte(tmpByteC2) || byte(0) || byte(0);

      drop tmpByte: _KMF_seq;
   run;
%end;

%* STAGE 2: CONSTRUCT ENTIRE LINES FOR KMF FILE ;
%* ------------------------------------------- ;
%if & KMG abort = N %then %do;
   data _KMG_lines / view=_KMG_lines;
      retain seqno ;
      length line1 $15 line2 $1 line3 $&_KMG_nameLen line4 $1 line5 $1 line6 $3 line7 $1
             line8 $32 line9 $&_KMG_instrLen line10 $3 line11 $1;
      set _KMG_prep;

      %* following lines contain fixed text ;
      retain line2 '3' line4 '' line5 '1' line6 '332' line7 '1' line11 '1';

      %* first line: number of characters (+ number of definition for first definition) ;
      if _N_ = 1 then line1 = 'KM' || byte(0) || byte(2) || noDefByte || noCharByte || '252';
                 else line1 =                                            noCharByte || '252';

      %* line three contains definition name ;
      line3 = strip(_KMF_name);

      %* line eight contains number of lines in insert string;
      line8 = strip(noLinesIS);

      %* line nine contains insert string ;
      line9 = strip(_KMF_inStr);

      %* second to last line contains definition sequence number ;
      line10 = strip(seqno);
      keep seqno line:;
   run;
%end;

%* STAGE 3: CREATE KMF FILE ;
%* ------------------------ ;
%if &_KMG_abort = N %then %do;
   filename _KMG_out "&outLoc_\&outFile_";

   %* note line4 is not output, since it is always an empty line ;
   data _null_;
      set _KMG_lines;
      file _KMG_out termstr=NL lrecl=32767;
      put line1;
      put line2;
      put line3;
      put ;
      put line5;
      put line6;
      put line7;
      put line8;
      put line9;
      put line10;
      put line11;
   run;

   filename _KMG_out ;
%end;

%* restore original setting ;
options &_KMG_compress;
```

```
   %if %upcase(&del_) = YES %then %do;
      proc datasets nolist;
         delete _KMG_: / memtype=VIEW;
      quit;
   %end;
%mend KMFgenerator;

libname kmf "X:\xxx\Phuse\2012\SGS paper\2. example code\3. kmfFiles" access=readonly;

%KMFgenerator(inDS_    = kmf._kmf_input,
              outLoc_  = X:\xxx\Phuse\2012\SGS paper\2. example code\3. kmfFiles,
              outFile_ = phuseExample.kmf);
```