Paper 053-2009

Variable Names Don't Begin with the Same Characters? No Problem: How to Create Variable List without Copying, Pasting or Excel Intervention

Xu Zeng, Independent Consultant, Fairfax, VA

ABSTRACT

Working with hundreds even thousands of variables is very common for many SAS programmers. As such, SAS has provided variable list tools like VAR1-VARN or VAR: so that we do not need to type variable names one by one. However, when the variable names do not begin with the same character string, a.k.a. prefix, the SAS default variable list will not work. This paper demonstrates a dynamic yet simple time-saver to generate variable lists when variable names have other commonalities than prefixes using PROC SQL SELECT INTO with DICTIONARY.COLUMNS table. This powerful and innovative method also enables the creation of new variable names, renaming of variables, and identification of common variables without copying, pasting or Excel intervention. In addition, the paper illustrates how to create dataset name list, which is not an inherent SAS tool. This presentation is based on SAS® version 9.1.3, is not limited to any particular operating system, and is intended for intermediate to advanced SAS programmers who are quite familiar with PROC SQL and SAS Macro.

INTRODUCTION

PROC SQL SELECT VAR1 INTO: MVAR1 creates a macro variable MVAR1 with the value of first observation of VAR1. The power comes from using PROC SQL SELECT VAR1 INTO: MVAR1 with SEPARATED BY keywords. PROC SQL SELECT VAR1 INTO: MVAR1 SEPARATED BY ' ' creates a macro variable MVAR1 with every single value of VAR1 separated by blanks. It is even more powerful when we use WHERE clause to select certain values of VAR1. The following creates a macro variable MVAR1 with every value of VAR1 that satisfies the WHERE condition separated by blanks.

```
PROC SQL;
SELECT VAR1 INTO: MVAR1 SEPARATED BY ' '
FROM TABLE1
WHERE VAR1 CONDITION;
QUIT;
```

Since we are going to create variable name lists, we need to select from a table that contains variable names. PROC CONTENTS will probably be our first reaction. However, PROC CONTENTS DATA=TABLE1 OUT=TABLE1VARS is not only an extra step, but it also gives us variables in only one table. How can we create a table that contains the variables in every single table? We do not need to. SAS has already created it for us! SAS DICTIONARY tables are special read-only PROC SQL tables. They retrieve information about all the SAS data libraries, SAS data sets, SAS system options, and external files that are associated with the current SAS session[1]. We can only access them through PROC SQL. There are total 21 tables and one of them, DICTIONARY.COLUMNS, contains information about variables and their attributes of all SAS data files in the current SAS session[1]. Issue command DESCRIBE TABLE DICTIONARY.COLUMNS in PROC SQL, then we can see its contents in the log. Below, column "name" stores variable names and "memname" stores SAS dataset names.

```
Create table DICTIONARY.COLUMNS
(
libname char(8) label='Library Name',
memname char(32) label='Member Name',
memtype char(8) label='Member Type',
name char(32) label='Column Name',
type char(4) label='Column Type',
```

[1] SAS® Online Doc 9.1.3 for the Web ->Base SAS®->SAS SQL Procedure User's Guide->Programming with the SQL Procedure-> Accessing SAS System Information Using DICTIONARY Tables

```
length num label='Column Length',
npos num label='Column Position',
varnum num label='Column Number in Table',
label char(256) label='Column Label',
format char(49) label='Column Format',
informat char(49) label='Column Informat',
idxusage char(9) label='Column Index Type',
sortedby num label='Order in Key Sequence',
xtype char(12) label='extended Type',
notnull char(3) label='Not NULL?',
precision num label='Precision',
scale num label='Scale',
transcode char(3) label='Transcoded?'
);
```

Combining PROC SQL SELECT INTO with DICTIONARY.COLUMNS, we get the basic syntax of the method:

```
PROC SQL;
SELECT NAME INTO: MYVARLIST SEPARATED BY ' '
FROM DICTIONARY.COLUMNS
WHERE LIBNAME='MYLIB'
AND   MEMNAME='MYDATASET'
AND   NAME CONDITION;
QUIT;
```

Now let's go through some examples to see how the method works.

VARIABLE NAMES END WITH THE SAME CHARACTER

During my work in anti-dumping programming, I worked with foreign manufacturers cost and sales files of products they sold in both their home country and the US. The Import Administration of Department of Commerce requires that all home country variable names end with an 'H' and US variable names end with a 'U'. Here are examples of some cost variables:

Description	Home country variable	US variable
Quantity	QTYH	QTYU
Moving cost	MOVEH	MOVEU
Inland freight	INLFTWH	INLFTWU
Packing cost	PACKH	PACKU
Warehouse cost	WAREHSH	WAREHSU

Except quantity, all other variables are per unit basis. Sometimes I had to manipulate just home country variables or US variables. For example, to calculate sums of all US variables weighted by quantity without typing the variables one by one or copying and pasting, I can do the following:

```
PROC SQL NOPRINT;
SELECT NAME INTO: USVARS SEPARATED BY ' '
FROM DICTIONARY.COLUMNS
WHERE UPCASE(LIBNAME)='WORK'
AND   UPCASE(MEMNAME)='COST'
AND   UPCASE(NAME) LIKE '%U'
AND   UPCASE(NAME) NE 'QTYU';
QUIT;
```

The above SQL procedure creates a macro variable USVARS. It contains all the variable names ending with 'U' except QTYU from WORK.COST, and they are separated by blanks. UPCASE is used to help character matching.

```
%LET USVARS=&USVARS;
```

The %LET allows me to see the value of &USVARS in log and also gets rid of trailing blanks. Here is what I see in the log file when I have SYMBOLGEN turned on:

```
SYMBOLGEN:  Macro variable USVARS resolves to MOVEU WAREHSU INLFTWU INSUREU
WARRU PACKU VCOMU TCOMU FURMANU
```

Then, I can use PROC MEANS to calculate the weighted sum of each US variable and output them into a new data set USWGT. The weighted sum variables have the same names as original cost variables:

```
PROC MEANS DATA=COST NOPRINT;
VAR &USVARS;
WEIGHT QTYU;
OUTPUT OUT=USWGT SUM=;
RUN;
```

Let's take the above example up a level. Suppose I have to add a prefix WGT_ to all weighted sum variables, i.e. weighted sum of MOVEU will be named WGT_MOVEU. Without any copying and pasting, here is what I can do:

```
PROC SQL NOPRINT;
SELECT COMPRESS('WGT_'||NAME) INTO: WGT_USVARS SEPARATED BY ' '
FROM DICTIONARY.COLUMNS
WHERE UPCASE(LIBNAME)='WORK'
AND   UPCASE(MEMNAME)='COST'
AND   UPCASE(NAME) LIKE '%U'
AND   UPCASE(NAME) NE 'QTYU';
QUIT;

%LET WGT_USVARS=&WGT_USVARS;
```

The log shows:

```
SYMBOLGEN:  Macro variable WGT_USVARS resolves to WGT_MOVEU WGT_WAREHSU
WGT_INLFTWU WGT_INSUREU WGT_WARRU WGT_PACKU WGT_VCOMU WGT_TCOMU WGT_FURMANU
```

Then I just simply add &WGT_USVARS after SUM=:

```
PROC MEANS DATA=COST NOPRINT;
VAR &USVARS;
WEIGHT QTYU;
OUTPUT OUT=USWGT SUM=&WGT_USVARS;
RUN;
```

Now in the USWGT dataset, weighted sum variable names have prefix 'WGT_'. When using SUM=NAMES in the OUTPUT statement of PROC MEANS, we must make sure that the output variable order is the same as that of the input variables in the VAR statement. This method guarantees the same order.

ADD ONE YEAR TO ALL THE DATE VARIABLES

Date variables are stored as numeric variables and usually have format associated with them. In the US sales dataset, each sales transaction is marked with several dates. Below are some examples:

Description	Name	Format
Sale invoice date	SALINDTU	date9.
Sale date	SALEDATU	date7.
Ship date	SHIPDATU	date7.
Pay date	PAYDATEU	yymmdd8.
Surcharge date	SURDATEU	mmddyy10.
Enter date	ENTDTU	mmddyy10.

Usually they assume one format; however, I changed them to different formats in order to show the programming flexibility.

Let's say there was a data preparation error, and all the dates are in year 2005 instead of 2006. I need to add one year to all the dates. Even though the above date variable names contain common text like 'DATE', 'DAT' or 'DT', I cannot use character matching to get all the right variables because in the same sales dataset, there are other variable names that contain text 'DT'. However, I can utilize the date format since non-date variables should not have one.

```
PROC SQL NOPRINT;
SELECT NAME INTO: DATEVARS SEPARATED BY ' '
FROM DICTIONARY.COLUMNS
WHERE UPCASE(LIBNAME)='WORK'
AND   UPCASE(MEMNAME)='USSALES'
AND   (UPCASE(FORMAT) CONTAINS 'MMDD'
OR     UPCASE(FORMAT) CONTAINS 'DATE'
OR     UPCASE(FORMAT) CONTAINS 'YYMM'
OR     UPCASE(FORMAT) CONTAINS 'DDMM');

/* C STANDS FOR CORRECTED */
SELECT COMPRESS('C_'||NAME) INTO: C_DATEVARS SEPARATED BY ' '
FROM DICTIONARY.COLUMNS
WHERE UPCASE(LIBNAME)='WORK'
AND   UPCASE(MEMNAME)='USSALES'
AND   (UPCASE(FORMAT) CONTAINS 'MMDD'
OR     UPCASE(FORMAT) CONTAINS 'DATE'
OR     UPCASE(FORMAT) CONTAINS 'YYMM'
OR     UPCASE(FORMAT) CONTAINS 'DDMM');
QUIT;
```

'MMDD','DATE','YYMM' and 'DDMM' cover most of the commonly used date formats.

```
%LET DATEVARS=&DATEVARS;
%LET C_DATEVARS=&C_DATEVARS;
```

Below is displayed in the log, and the two variable lists have the same variable order:

```
SYMBOLGEN:  Macro variable DATEVARS resolves to SALINDTU SALEDATU SHIPDATU
PAYDATEU SURDATEU ENTDTU

SYMBOLGEN:  Macro variable C_DATEVARS resolves to C_SALINDTU C_SALEDATU
C_SHIPDATU C_PAYDATEU C_SURDATEU C_ENTDTU
```

Now I can correct the date variables as the following:

```
DATA USSALES(DROP=I);
SET USSALES;
ARRAY WRONGDATE[*] &DATEVARS;
ARRAY CORRECTDATE[*] &C_DATEVARS;
DO I=1 TO DIM(WRONGDATE);
   CORRECTDATE[I]=WRONGDATE[I]+365;
END;
RUN;
```

RENAMING VARIABLES

When I was at Capital One, I worked with monthly credit bureau files of credit card accounts. Each monthly file contained 400-500 variables. Sometimes, I had to merge multiple monthly files together by account number and

analyze variable trend. Since the variable names were the same, I had to rename them before the merge. For example, I renamed variable P1 to P1_1 for month 1, P1_2 for month 2, etc. Sometimes I used Excel to create rename equations. It turns out that the same PROC SQL SELECT INTO can generate all the rename equations in one simple step. No more Excel intervention is needed.

Suppose I have three monthly files I need to merge together by ACCTNO. Each file has 464 variables besides ACCTNO. I will add suffix _1, _2 and _3 to them respectively. Below is the complete code:

```
%MACRO RN(FILE,SUFFIX);

PROC SQL NOPRINT;
SELECT COMPRESS(NAME||"="||NAME||"_&SUFFIX") INTO: RENAME_&SUFFIX
SEPARATED BY ' '
FROM DICTIONARY.COLUMNS
WHERE UPCASE(LIBNAME)='WORK'
AND   UPCASE(MEMNAME)="&FILE"
AND   UPCASE(NAME) NE 'ACCTNO';
QUIT;

PROC DATASETS LIB=WORK NOLIST NODETAILS;
MODIFY &FILE;
RENAME &&RENAME_&SUFFIX;
QUIT;

%MEND;

%RN(MONTH1,1);
%RN(MONTH2,2);
%RN(MONTH3,3);
```

In SELECT statement, COMPRESS(NAME||"="||NAME||"_&SUFFIX") constructs the rename equations. For example, variable P1 of month 1 file will have P1=P1_1. All the equations are separated by blanks and are stored in macro variables RENAME_1, RENAME_2 and RENAME_3. Of course, I have to exclude ACCTNO since it is the merge by variable.

Below is value of RENAME_1 displayed in the log:

```
SYMBOLGEN:  Macro variable RENAME_1 resolves to P1=P1_1 P2=P2_1 P3=P3_1 P4=P4_1
P5=P5_1 P6=P6_1 P7=P7_1 P8=P8_1 P9=P9_1 P10=P10_1 P11=P11_1 P12=P12_1 P13=P13_1
P14=P14_1 P15=P15_1 P16=P16_1 P17A=P17A_1 P17B=P17B_1 P18A=P18A_1 P18B=P18B_1
P19=P19_1 P20=P20_1 P21=P21_1 P22=P22_1 P23=P23_1 P24=P24_1 P25=P25_1 P26=P26_1
P27=P27_1 P28=P28_1 P29=P29_1 P30=P30_1 P31=P31_1 P32=P32_1 P33=P33_1 P34=P34_1
P35=P35_1 P36A=P36A_1 P36B=P36B_1 P36C=P36C_1 P36D=P36D_1 P37A=P37A_1 P37B=P37B_1
P37C=P37C_1 P37D=P37D_1 P38A=P38A_1 P38B=P38B_1 P38C=P38C_1 P38D=P38D_1 P38E=P38E_1
P38F=P38F_1 P39=P39_1 P40=P40_1 P41A=P41A_1 P41B=P41B_1 P42=P42_1 E1=E1_1 E2=E2_1
E3=E3_1 E4=E4_1 E5=E5_1 E6=E6_1 E7A1=E7A1_1 E7A2=E7A2_1 E7A3=E7A3_1 E7A4=E7A4_1
E7A5=E7A5_1 E7A6=E7A6_1 E7A7=E7A7_1 E7B1=E7B1_1 E7B2=E7B2_1 E7B3=E7B3_1 E7B4=E7B4_1
E7B5=E7B5_1 E7B6=E7B6_1 E7B7=E7B7_1 E8=E8_1 E9=E9_1 E10=E10_1 E11=E11_1 E12=E12_1
E13=E13_1 E14=E14_1 E15=E15_1 E16=E16_1 E17=E17_1 E18=E18_1 E19=E19_1 E20=E20_1
E21=E21_1 E22=E22_1 E23=E23_1 E24=E24_1 E25=E25_1 E26=E26_1 E27=E27_1 E28=E28_1
E29=E29_1 E30=E30_1 E31=E31_1 E32=E32_1 E33=E33_1 E34=E34_1 E35=E35_1 E36=E36_1
E37=E37_1 E38=E38_1 E39=E39_1 E40=E40_1 E41=E41_1 E42A=E42A_1 E42B=E42B_1 E42C=E42C_1
E43A=E43A_1 E43B=E43B_1 E43C=E43C_1 E44A=E44A_1 E44B=E44B_1 E44C=E44C_1 E45A=E45A_1
E45B=E45B_1 E45C=E45C_1 E46A=E46A_1 E46B=E46B_1 E46C=E46C_1 E47A=E47A_1 E47B=E47B_1
E47C=E47C_1 E48=E48_1 E49=E49_1 E50=E50_1 E51=E51_1 E52=E52_1 E54=E54_1 SC7=SC7_1
SC8=SC8_1 SC9=SC9_1 SC10=SC10_1 SC11=SC11_1 SC12=SC12_1 SC13=SC13_1 SC14=SC14_1
SC15=SC15_1 SC16=SC16_1 SC17=SC17_1 SC18=SC18_1 SC19=SC19_1 SC20=SC20_1 SC21=SC21_1
SC22=SC22_1 SC23=SC23_1 SC24=SC24_1 SC25=SC25_1 SC26=SC26_1 SC27=SC27_1 SC28=SC28_1
SC29=SC29_1 SC30=SC30_1 SC31=SC31_1 SC32=SC32_1 SC33=SC33_1 SC34=SC34_1 SC35=SC35_1
```

```
SC36=SC36_1 SC37=SC37_1 SC38=SC38_1 SC39=SC39_1 SC40=SC40_1 SC41=SC41_1 SC42=SC42_1
SL1=SL1_1 SL2=SL2_1 SL3=SL3_1 SL4=SL4_1 SL5=SL5_1 SL6=SL6_1 SL7=SL7_1 SL8=SL8_1
SL9=SL9_1 SL10=SL10_1 SL11=SL11_1 AM6A1=AM6A1_1 AM6A2=AM6A2_1 AM6A3=AM6A3_1
AM6A4=AM6A4_1 AM6A5=AM6A5_1 AM6A6=AM6A6_1 AM6A7=AM6A7_1 AM6B1=AM6B1_1 AM6B2=AM6B2_1
AM6B3=AM6B3_1 AM6B4=AM6B4_1 AM6B5=AM6B5_1 AM6B6=AM6B6_1 AM6B7=AM6B7_1 AM7=AM7_1
AM8=AM8_1 AM9=AM9_1 AM10=AM10_1 AM11=AM11_1 AM12=AM12_1 AM13=AM13_1 AM14=AM14_1
AM15=AM15_1 AM16=AM16_1 AM17=AM17_1 AM18A=AM18A_1 AM18B=AM18B_1 AM19A=AM19A_1
AM19B=AM19B_1 AM20=AM20_1 AM21A=AM21A_1 AM21B=AM21B_1 AM21C=AM21C_1 AM21D=AM21D_1
AM21E=AM21E_1 AM21F=AM21F_1 AM21G=AM21G_1 AM22A=AM22A_1 AM22B=AM22B_1 AM22C=AM22C_1
AM22D=AM22D_1 AM22E=AM22E_1 AM22F=AM22F_1 AM22G=AM22G_1 AM23A=AM23A_1 AM23B=AM23B_1
AM24=AM24_1 AM25=AM25_1 AM26=AM26_1 AM27A=AM27A_1 AM27B=AM27B_1 AM27C=AM27C_1
AM27D=AM27D_1 AM28A=AM28A_1 AM28B=AM28B_1 AM28C=AM28C_1 AM28D=AM28D_1 AM29=AM29_1
AM30=AM30_1 AM31=AM31_1 AM32A=AM32A_1 AM32B=AM32B_1 AM32C=AM32C_1 AM32D=AM32D_1
AM32E=AM32E_1 AM32F=AM32F_1 AM33A=AM33A_1 AM33B=AM33B_1 AM33C=AM33C_1 AM33D=AM33D_1
AM33E=AM33E_1 AM33F=AM33F_1 AM34A=AM34A_1 AM34B=AM34B_1 AM34C=AM34C_1 AM34D=AM34D_1
AM34E=AM34E_1 AM34F=AM34F_1 AM35A=AM35A_1 AM35B=AM35B_1 AM35C=AM35C_1 AM35D=AM35D_1
AM35E=AM35E_1 AM35F=AM35F_1 AM35G=AM35G_1 AM36A=AM36A_1 AM36B=AM36B_1 AM36C=AM36C_1
AM36D=AM36D_1 AM36E=AM36E_1 AM36F=AM36F_1 AM36G=AM36G_1 AM37A=AM37A_1 AM37B=AM37B_1
AM37C=AM37C_1 AM38=AM38_1 AM39=AM39_1 AM40=AM40_1 AM41A=AM41A_1 AM41B=AM41B_1
AM42A=AM42A_1 AM42B=AM42B_1 AM42C=AM42C_1 AM43A=AM43A_1 AM43B=AM43B_1 AM43C=AM43C_1
RM1A=RM1A_1 RM1B=RM1B_1 RM1C=RM1C_1 RM1D=RM1D_1 RM2A=RM2A_1 RM2B=RM2B_1 RM2C=RM2C_1
RM2D=RM2D_1 RM3A=RM3A_1 RM3B=RM3B_1 RM3C=RM3C_1 RM3D=RM3D_1 RM4A=RM4A_1 RM4B=RM4B_1
RM4C=RM4C_1 RM4D=RM4D_1 RM5A=RM5A_1 RM5B=RM5B_1 RM5C=RM5C_1 RM5D=RM5D_1 RM6=RM6_1
RM7=RM7_1 RM8A=RM8A_1 RM8B=RM8B_1 RM8C=RM8C_1 RM9=RM9_1 RM10=RM10_1 RM11=RM11_1
RM12=RM12_1 RM13=RM13_1 RM14=RM14_1 RM15=RM15_1 RM16=RM16_1 RM17=RM17_1 RM18=RM18_1
A1A=A1A_1 A1B=A1B_1 A1C=A1C_1 A1D=A1D_1 A1E=A1E_1 A1F=A1F_1 A2A=A2A_1 A2B=A2B_1
A2C=A2C_1 A2D=A2D_1 A2E=A2E_1 A2F=A2F_1 A3A=A3A_1 A3B=A3B_1 A3C=A3C_1 A3D=A3D_1
A3E=A3E_1 A3F=A3F_1 CS11A=CS11A_1 CS11B=CS11B_1 CS12A=CS12A_1 CS12B=CS12B_1
REC01=REC01_1 REC02=REC02_1 REC03=REC03_1 REC04=REC04_1 REC05A=REC05A_1
REC05B=REC05B_1 REC05C=REC05C_1 REC06A=REC06A_1 REC06B=REC06B_1 REC06C=REC06C_1
REC07=REC07_1 REC08A=REC08A_1 REC08B=REC08B_1 REC09=REC09_1 REC10=REC10_1
REC11=REC11_1 REC12=REC12_1 MTG01=MTG01_1 TRIG16B=TRIG16B_1 TRIG33B=TRIG33B_1
TRIG34B=TRIG34B_1 TRIG35B=TRIG35B_1 TRIG37B=TRIG37B_1 MTG02=MTG02_1 MTG03=MTG03_1
MTG04=MTG04_1 MTG05=MTG05_1 MTG06=MTG06_1 REC13=REC13_1 AM44=AM44_1 REC14=REC14_1
REC15=REC15_1 REC16=REC16_1 REC17=REC17_1 REC18=REC18_1 REC19=REC19_1 MTG07=MTG07_1
MTG08=MTG08_1 MTG09=MTG09_1 MTG10=MTG10_1 MTG11=MTG11_1 SPAM01=SPAM01_1
SPAM02=SPAM02_1 SPAM03=SPAM03_1 SPAM04=SPAM04_1 SPAM05=SPAM05_1 SPAM06=SPAM06_1
SPAM07=SPAM07_1 SPAM08=SPAM08_1 SPAM09=SPAM09_1 SPAM10=SPAM10_1 SPAM11=SPAM11_1
SPAM12=SPAM12_1 SPAM13=SPAM13_1 SPAM14=SPAM14_1 SPAM15=SPAM15_1 SPAM16=SPAM16_1
SPAM17=SPAM17_1 SPAM18=SPAM18_1 SPAM19=SPAM19_1 SPAM20=SPAM20_1 SPAM21=SPAM21_1
SPAM22=SPAM22_1 SPAM23=SPAM23_1 SPAM24=SPAM24_1 SPAM25=SPAM25_1 SPAM26=SPAM26_1
SPAM27=SPAM27_1 SPAM28=SPAM28_1 SPAM29=SPAM29_1 SPAM30=SPAM30_1 SPAM31=SPAM31_1
SPAM32=SPAM32_1 SPAM33=SPAM33_1 SPAM34=SPAM34_1 SPAM35=SPAM35_1 SPAM36=SPAM36_1
SPAM37=SPAM37_1 SPAM38=SPAM38_1 SPAM39=SPAM39_1 SPAM40=SPAM40_1 SPAM41=SPAM41_1
AUTO1=AUTO1_1 AUTO2=AUTO2_1 AUTO3=AUTO3_1 AUTO4=AUTO4_1 AUTO5=AUTO5_1 AUTO6=AUTO6_1
AUTO7=AUTO7_1 AUTO8=AUTO8_1 AUTO9=AUTO9_1 AUTO10=AUTO10_1 AUTO11=AUTO11_1
AUTO12=AUTO12_1 AUTO13=AUTO13_1 AUTO14=AUTO14_1 AUTO15=AUTO15_1 AUTO16=AUTO16_1
AUTO17=AUTO17_1 AUTO18=AUTO18_1 AUTO19=AUTO19_1 AUTO20=AUTO20_1 AUTO21=AUTO21_1
AUTO22=AUTO22_1 AUTO23=AUTO23_1 AUTO24=AUTO24_1 AUTO25=AUTO25_1 AUTO26=AUTO26_1
AUTO27=AUTO27_1 AUTO28=AUTO28_1 AUTO29=AUTO29_1 AUTO30=AUTO30_1 AUTO31=AUTO31_1
AUTO32=AUTO32_1 AUTO33=AUTO33_1 AUTO34=AUTO34_1 AUTO35=AUTO35_1 SC4=SC4_1 SC5=SC5_1
SC6=SC6_1
```

Is this too much text for one macro variable? No. SAS Macro variable can store up to 65,534 characters[2], which is quite a large number. In the above example, RENAME_1, RENAME_2 and RENAME_3 each has 5871 characters, which is less than 10% of the maximum. This is very powerful. We can use it to generate other SAS statements on thousands of variables.

[2] SAS® Online Doc 9.1.3 for the Web ->Base SAS®->SAS Macro Language-> Understanding and Using the Macro Facility->Macro Variables-> Macro Variables Defined by Users.

IDENTIFYING COMMON VARIABLES IN DIFFERENT DATASETS

At my current consulting job, I often have to compare two related datasets and identify common variables. In one simple step, the following code identifies and stores common variables of two datasets in a macro variable, which I can refer to in later programming:

```
PROC SQL NOPRINT;
SELECT A.NAME INTO: SAMEVARS SEPARATED BY ' '
FROM
(SELECT NAME
 FROM DICTIONARY.COLUMNS
 WHERE UPCASE(LIBNAME)='XU'
 AND UPCASE(MEMNAME)='DATASET1') A,
(SELECT NAME
 FROM DICTIONARY.COLUMNS
 WHERE UPCASE(LIBNAME)='XU'
 AND UPCASE(MEMNAME)='DATASET2') B
WHERE A.NAME=B.NAME;
QUIT;
```

To identify the variables in the first dataset but not in the second, I can simply replace the inner join with the EXCEPT union:

```
PROC SQL NOPRINT;
SELECT NAME INTO: FIRSTVARS SEPARATED BY ' '
FROM
(
 SELECT NAME
 FROM DICTIONARY.COLUMNS
 WHERE UPCASE(LIBNAME)='XU'
 AND UPCASE(MEMNAME)='DATASET1'
 EXCEPT
 SELECT NAME
 FROM DICTIONARY.COLUMNS
 WHERE UPCASE(LIBNAME)='XU'
 AND UPCASE(MEMNAME)='DATASET2'
);
QUIT;
```

CREATE SAS DATASET NAME LIST

SAS dataset name list is not directly supported by SAS, i.e. we cannot use DATA1-DATAN in SET or MERGE statement. However, we can create our own using PROC SQL SELECT INTO with DICTIONARY.TABLES. As I mentioned earlier, there are 21 DICTIONARY tables. DICTIONARY.TABLES or DICTIONARY.MEMBERS can be used to manage current session SAS tables/views and SAS files[3]. Here are some important columns in DICTIONARY.TABLES:

```
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type'
   dbms_memtype char(32) label='DBMS Member Type',
   memlabel char(256) label='Dataset Label',
   typemem char(8) label='Dataset Type',
   crdate num format=DATETIME informat=DATETIME label='Date Created',
   modate num format=DATETIME informat=DATETIME label='Date Modified',
```

[3] SAS® Online Doc 9.1.3 for the Web ->Base SAS®->SAS SQL Procedure User's Guide->Programming with the SQL Procedure-> Accessing SAS System Information Using DICTIONARY Tables

```
   nobs num label='Number of Physical Observations',
   obslen num label='Observation Length',
   nvar num label='Number of Variables'
```

At my current consulting job, a statistical model score dataset named SCORE_MMDDYY is generated daily. Sometimes, I need to set multiple days of the score datasets together. The following code creates a macro variable SCORESDS. It contains the names of datasets that were created between DATE1 and DATE2 separated by blanks.

```
PROC SQL NOPRINT;
SELECT MEMNAME INTO: SCORESDS SEPARATED BY ' '
FROM DICTIONARY.TABLES
WHERE LIBNAME='SCORE'
AND UPCASE(MEMNAME) LIKE 'SCORE%'
AND DATEPART(CRDATE) BETWEEN DATE1 AND DATE2;
QUIT;
```

Then, I can simply call the macro variable in SET statement as follows:

```
DATA ALLSCORES;
SET &SCORESDS;
RUN;
```

CONCLUSION

As my paper shows, PROC SQL SELECT INTO with SAS DICTIONARY.COLUMNS table is an amazing tool. It allows us to create custom variable lists, create new variable names, rename variables, and identify common variables without copying, pasting or Excel intervention. Furthermore, with DICTIONARY.TABLES, we can create dataset name list, which is not directly supported by SAS. Using this method since I first discovered it in 2005, I have dramatically saved time and increased my productivity. I hope I have brought forward a valuable tool that you can utilize and adapt for your work.

REFERENCES

SAS Institute Inc. “SAS Macro Language” “SAS SQL Procedure User’s Guide” SAS® OnlineDoc 9.1.3 for the Web <http://support.sas.com/onlinedoc/913/docMainpage.jsp>

CONTACT INFORMATION

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

Xu Zeng
3012 Hickory Grove Court
Fairfax, VA 22031
703-864-8914
xuzeng@hotmail.com