

CC01

Putting SAS Dataset Variable Names into a Macro Variable

Stephanie R. Thompson, University of Memphis, Memphis, TN

ABSTRACT

Have you ever wanted to use the variable names from a SAS dataset in a macro variable? Maybe you need all or just some of the variable names to be useable in keep statements, select statements, or some other parts of your program. There are several ways to obtain a listing of the variable names in a SAS dataset and three options will be compared and contrasted: DICTONARY tables, SASHELP.VCOLUMNS, and PROC CONTENTS.

Some people prefer to use DICTONARY tables rather than process the dataset headers through PROC CONTENTS. However, use of the DICTONARY tables or the related SASHELP.VCOLUMN view gives rise to problems of case sensitivity and variable specification that can be easily handled by using PROC CONTENTS to process the header information. When using PROC CONTENTS to output variable names in a dataset, we need to take care that we are not affecting other processes. This paper will look at methods to assure that this doesn't happen.

INTRODUCTION

There are many times when a list of variable names might be needed in a SAS program. Keep statements, select statements, and print procedures are some examples that come to mind. Typing in a long list of variable names can be tedious and result in typographical errors. Programmatically reading in the variable names eliminates these problems. DICTONARY tables, the SASHELP.VCOLUMN view, or PROC CONTENTS are options. Each method to read in the variable names will be reviewed first. Secondly, the code to insert the variable names into a macro variable – delimited or not – will be covered. This section of code is also adaptable to only select certain variable names.

COMPARING METHODS

This section compares different aspects of each method. The overview will provide information on how SAS processes the code and where the data sources are. General and example specific syntax will be provided to illustrate how much or how little code is needed. Finally, system performance and other processing issues will be discussed.

In order to compare methods, the dataset DEGREE05.SAS7BDAT was used. This dataset contains 3467 observations and 61 variables and it contains information about students receiving a degree in academic year 2005. The variable MAJOR currently has a length of 20 but it needs to be increased to 25 characters in order to accommodate a new major recently introduced by the university.

DICTONARY TABLES

Dictionary tables are set up automatically in SAS and they retrieve information about all SAS datasets, libraries, system options, and external files for the current SAS sessions. These tables can only be accessed with PROC SQL. DICTONARY.COLUMNS contains information on columns and their attributes. To see how a table is structured, you can type:

```
proc sql;
    describe table dictionary.columns;
quit;
```

In order to read in the variable name and their position for that example dataset the code below would be used:

```
proc sql;
    create table vars as
    select varnum, name
    from dictionary.columns
    where memname = 'DEGREE05';
quit;
```

It is important that the dataset name that you are referring to be in upper case. DICTONARY tables store member names and libnames in upper case. It is also worthwhile to note that DICTONARY tables are read-only and therefore cannot be modified.

SASHELP.VCOLUMN

The SASHELP views can be used in the DATA step and procedures. They are not limited to just PROC SQL. To retrieve variable number and name using SASHELP.VCOLUMNS, this code can be used:

```
data vars;
set sashelp.vcolumn;
where memname = 'DEGREE05';
keep varnum name;
run;
```

PROC CONTENTS

Using PROC CONTENTS allows you to use a KEEP= dataset option to specify variables without regard to case sensitivity. It also allows you to use the existing and very handy SAS list features such as hyphens, double hyphens, colons, -NUMERIC-, and -CHARACTER-. This option can be more flexible than either DICTIONARY.TABLES or SASHELP.VCOLUMN. The syntax is shown below.

```
proc contents data = libname.DEGREE05
    out = vars(keep = varnum name)
    noprint;
run;
```

PROCESSING ISSUES

Just for comparison purposes, each of the methods was run on SAS 9.1.3 with Service Pack 4 on a Dell Optiplex GX150 to see if there was any difference in performance. Each method was run four times. The table below summarizes the results average times in seconds with the range in parentheses.

Method	Real time from Log	CPU time from Log
proc contents	0.11 (0.18 – 0.06)	0.04 (0.06-0.03)
DICTIONARY.COLUMNS	0.33 (1.07 – 0.07)	0.13 (0.24-0.04)
SASHELP.VCOLUMN	6.61 (10.47 – 10.47)	2.67 (3.57 – 2.27)

Using SASHELP.VCOLUMN in a DATA step requires the DICTIONARY.COLUMNS dataset to be read in which explains its longer processing time. For this example, proc contents and DICTIONARY tables are quite comparable in performance. The difference in processing times is not an issue with such a small dataset. However, if you need to access many tables or large tables speed may become a factor.

CREATING THE MACRO VARIABLE

Once the variable numbers and names have been read, you can then use them to create a macro variable. The code to accomplish putting the variable names into a macro variable is a 2 step process. The first step uses PROC CONTENTS and the second PROC SQL. A space is used in this instance to separate the variable names. The code is shown below:

```
proc contents data = libname.DEGREE05
    out = vars(keep = varnum name)
    noprint;
run;

proc sql noprint;
    select distinct name
    into :orderedvars separated by ' '
    from vars
    order by varnum;
quit;
```

Issuing a subsequent %PUT statement will yield this in the log:

```
223 %put &orderedvars;
    awd_term system inst loc ... continues to list all 61 variables
```

Here, the put statement shows all variable names ordered by the variable number in the original SAS dataset.

SELECTING ONLY SOME VARIABLES

There may be occasion to select only some of the variables in a dataset into a macro variable. Maybe you want to drop or keep some variables or use them separated by a delimiter. The code from above was modified to show a sub selection of variables with a different delimiter. Two different methods are shown. The first uses the variable number and the second a KEEP= option.

USING VARIABLE NUMBER

```
proc sql noprint;
    select distinct name
    into :orderedvars2 separated by ', '
    from vars
    where varnum in (11, 12, 18)
    order by varnum;
quit;
```

In this case the %PUT results in:

```
234 %put &orderedvars2;
degree, major_1, resident
```

KEEP= OPTION

```
proc contents data=libname.DEGREE05 (keep= degree major_1 resident)
    out=orderedvars3 (keep= varnum name)
    noprint;
run;
```

```
proc sql noprint ;
    select name into :orderedvars3 separated by ' and '
    from vars
    order by varnum;
quit;
```

Now the %PUT results in:

```
563 %put &orderedvars2;
degree and major_1 and resident
```

DELIMITER VARIATIONS

Any set of characters can be used for the delimiter or what goes between the single quotes in the separated by line of code. You can insert something creative such as '___**' which might be useful for printing in a title or report header. Different character combinations can be used for different purposes. The "and" can be used in conditional logic and the ", " in a SELECT statement in PROC SQL.

Another delimiter that's useful when writing out a tab-delimited text file is the tab. It's useful to start the output file with a tab-delimited row of variable names. This can be done using the following in the code:

```
select name into: tablist separated by '09'x
```

CHANGING VARIABLE LENGTH

Now suppose we want to change the length of the variable major_1 from 20 to 25 characters. The following code illustrates how this can easily be done with PROC CONTENTS.

```
proc contents data=libname.DEGREE05 (keep= degree major_1 resident gender level conc)
    out=orderedvars3
    noprint;
run;
```

```

proc sql noprint ;
    select name into :orderedvars3 separated by ' '
    from vars
    order by varnum;
quit;

data newdegree;
    retain &orderedvars3;
    length major_1 $25;
set libname.DEGREE05;
run;

```

ONE WARNING

When the SQL is executed, the following warning is shown in the log:

```

WARNING: The query as specified involved ordering by an item that doesn't appear
         in its SELECT clause. Since you are ordering the output of a SELECT
         DISTINCT it may appear that some duplicated have not been eliminated.

```

This warning has not caused any issues when I have used this code. My results have been as expected.

CONCLUSION

The goal of this paper was to present a way in which to read variable names into a macro variable with some type of delimiter. This macro variable can be used throughout your program and can serve many functions. Hopefully you will find this approach useful and that it will make your code more flexible and easier to write.

CONTACT INFORMATION

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

Stephanie R. Thompson
 University of Memphis
 411 Administration Bldg.
 Memphis, TN 38152-3370
 Work Phone: (901)678-5529
 Fax: (901) 678-5138
 Email: srthmpsn@memphis.edu
 Web: <http://oir.memphis.edu>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.