Paper D13-2009

Using the Data Step's ATTRIB Statement to both Manage and Document Variables in a SAS® Dataset (*lightly*)

Philip A. Wright, The University of Michigan, Ann Arbor, MI

ABSTRACT

There are several different ways in which to order variables in a SAS dataset. Some of them are quite similar, one is dangerous, and each of them must be used prior to a *set, merge,* or *update* statement. One of these statements—the *attrib* statement—can serve the roll of several other statements, is not dangerous, and can actually serve as *light* documentation in data step code. Generating a complete *attrib* statement for a large data set, however, can be a daunting task. The use of a macro that generates a complete *attrib* statement for a previously-generated data set can be of great use to manage the data set while the data step is further developed. Using the macro to generate a complete *attrib* statement and subsequently using the generated *attrib* statement to order/re-order and aid the conversion of variable types will be demonstrated.

INTRODUCTION

Although not always necessary for SAS processing, the order of variables in a dataset can be quite important when the variables or values in a dataset are reviewed by fellow programmers and supervisors. One standard for the ordering of variables might be the importance of variables within the dataset with regard to the domain the data represent—it might be more important to have some variables precede other variables. Another need to order variables may be based on grouping similar variables, a series of variables, or dependent variables.

Variables become 'ordered' in a dataset as they are defined in the Program Data Vector (PDV) by the compilation of data step code by the SAS compiler—the first variables initialized by the data step as parsed by the compiler are the first variables in the PDV. There are several different data step statements that can be used to order variables in the dataset(s) generated by the data step: *attrib, array, format, informat, length,* and *retain* (Source: SAS Knowledge Base/Samples & SAS Notes). Using any of these statements prior to a *set, merge,* or *update* statement will initialize the variables they specify in the PDV according to the order they are included in the statement.

In addition to ordering variables, other dataset management tasks include adding or changing variable labels, adding or changing format and informat specifications, changing the allocation of memory for variables, and converting variables of one type to the other.

Using a detailed and patterned version of the *attrib* statement at the beginning of data step code can order the variables in the dataset and serve as light documentation. Generating this statement with an editor could well be quite formidable, especially for large and complex datasets. The generation of the statement using several macros at the *end* of initial and iterative data step development and the subsequent insertion of the statement at the beginning of the data step is comparatively easy and can be quite useful for managing most datasets.

MANAGING VARIABLES IN A SAS DATASET

In addition to ordering variables, the data step programmer may well need to specify the labels, formats, informats, lengths, and transcode values for each variable if the default values are not suitable. I consider both the ordering of variables and the specification of variable attribute values as managing variables in a dataset.

Dependent upon the version of SAS you are using, there are five or six variable attributes whose values can be easily specified by the programmer: *name, type, length, format, informat,* and, in version 9, *transcode*. Every variable must have a name, a type specification (numeric or character) and a byte length specification (generally 1 to 32,767 bytes for character variables and 3 to 8 bytes for numeric variables). The SAS compiler assigns default values to the latter two of these attributes when these values are not explicitly specified; specifying values for the other variable attributes is not strictly necessary. Adding values for the *label* attribute, however, can add immeasurably to the ease of comprehension of a dataset and is virtually required for larger, more complex datasets. Assigning values to the attributes for each variable in the dataset using an *attrib* statement generated by macro code after each iterative development of the data step can make the management of the variables in the dataset comparatively light.

A second aspect of managing a dataset lightly is doing so with the fewest SAS resources as possible—not using a second data step to reorder variables or using *proc datasets* to change the values of variable attributes. The resources required for the processing of a secondary data step or procedure might be trivial for smaller datasets but could be quite significant for larger datasets.

Sometimes, however, a secondary data step might be preferable or even necessary. This can be particularly true where you must use a data step after a *proc SQL* join; there are some processing routines which are much easier to code in the data step and some processing routines that simply cannot be done in a single *proc SQL* join.

THE ATTRIB STATEMENT VERSUS OTHER STATEMENTS

The *attrib* statement can perform the function of the *format, informat, label,* and *length* statements and can specify the *transcode* value for each variable in the dataset (Source: SAS Knowledge Base/Samples & SAS Notes). Any combination of these attributes for any variable may be specified using the *attrib* statement. One catch common with using any of these statements to reorder variables is that they only reorder the variables that are included in the statement; the named variables consequently precede all other variables in the dataset. If you are working with a large dataset and want to reorder only the variables towards the end of the dataset then you must include all preceding variables in the statement. Although you may not need to specify attribute values for all variables in the dataset during early development of data step code, being able to readily change these values either individually, in groups, or globally during subsequent iterations of code development is quite useful.

A patterned layout of these specifications allows for easy changes, especially with the use of regular expressions, editing scripts, or an editor that supports the use of editing macros:

```
attrib
   date          label = 'Order Date'              length =   8
   sale          label = 'Unit Sale'               length =   8
   price         label = 'Unit Price'              length =   8
   discount      label = 'Price Discount'          length =   8
   cost          label = 'Unit Cost'               length =   8
   price1        label = 'Product 1 Unit Price'    length =   8
   price2        label = 'Product 2 Unit Price'    length =   8
   price3        label = 'Product 3 Unit Price'    length =   8
   price4        label = 'Product 4 Unit Price'    length =   8
   price5        label = 'Product 5 Unit Price'    length =   8
   price6        label = 'Product 6 Unit Price'    length =   8
   price7        label = 'Product 7 Unit Price'    length =   8
   price8        label = 'Product 8 Unit Price'    length =   8
   price9        label = 'Product 9 Unit Price'    length =   8
   price10       label = 'Product 10 Unit Price'   length =   8
   price11       label = 'Product 11 Unit Price'   length =   8
   price12       label = 'Product 12 Unit Price'   length =   8
   price13       label = 'Product 13 Unit Price'   length =   8
   price14       label = 'Product 14 Unit Price'   length =   8
   price15       label = 'Product 15 Unit Price'   length =   8
   price16       label = 'Product 16 Unit Price'   length =   8
   price17       label = 'Product 17 Unit Price'   length =   8
   regionName    label = 'Sales Region'            length = $ 7
   productLine   label = 'Name of product line'    length = $ 5
   productName   label = 'Product Name'            length = $ 9
   region        label = 'Region ID'               length =   8   format = 6.
   line          label = 'Product Line ID'         length =   8   format = 6.
   product       label = 'Product ID'              length =   8   format = 6.
;
```

Many people use the *retain* statement for ordering variables because only the variable names are needed for the statement—no other variable attribute specifications are required. Use of the *retain* statement, however, can be dangerous in subsequent iterations of code development as the intended function of the statement is to save values for the specified variables from one observation to the next. The danger arises when the values of any of the variables get conditionally assigned by the data step code and are not overwritten by an input statement; the conditional assignment may create a value that should not be retained for subsequent observations. Even if the data step programmer is careful not to do this, this mistake could be made should less knowledgeable programmers subsequently modify the code.

EXAMPLES

REORDERING VARIABLES WITH THE ATTRIB STATEMENT

1. The first step for using an *attrib* statement to reorder variables is to generate the statement itself by passing the two level name of the existing dataset for which you want to reorder the variables in a call to the macro code:

```
%list_attrib(ds_spec = work.pricedata) ;
```

(the macro code is available from sascommunity.org)

2. Once an *attrib* statement has been generated with macro code, it must be inserted using your code editor prior to any *set, merge,* or *update* statements in the data step code that generated the version of the dataset passed to the macro code:

BEFORE EDITING:

```
data work.pricedata ;
set sashelp.pricedata ;
run ;
```

AFTER EDITING:

```
data work.pricedata ;
attrib
   date          label = 'Order Date'              length =   8
   sale          label = 'Unit Sale'               length =   8
   price         label = 'Unit Price'              length =   8
   discount      label = 'Price Discount'          length =   8
   cost          label = 'Unit Cost'               length =   8
   price1        label = 'Product 1 Unit Price'    length =   8
   price2        label = 'Product 2 Unit Price'    length =   8
   price3        label = 'Product 3 Unit Price'    length =   8
;
set sashelp.pricedata ;
run ;
```

3. Once pasted, the programmer is then able to reorder the records for each variable as desired with the editor:

BEFORE REORDERING:

```
data work.pricedata ;
attrib
   date          label = 'Ord …
   sale          label = 'Uni …
   price         label = 'Uni …
   discount      label = 'Pri …
   cost          label = 'Uni …
   price1        label = 'Pro …
   price2        label = 'Pro …
   price3        label = 'Pro …
;
set sashelp.pricedata ;
run ;
```

AFTER REORDERING:

```
data work.pricedata ;
attrib
   date          label = 'Ord …
   sale          label = 'Uni …
   price         label = 'Uni …
   price1        label = 'Pro …
   price2        label = 'Pro …
   price3        label = 'Pro …
   discount      label = 'Pri …
   cost          label = 'Uni …
;
set sashelp.pricedata ;
run ;
```

The variables will now be ordered as desired once the dataset is regenerated by submitting the revised data step code to the processor.

This is a rather trivial example. The real power of this technique becomes apparent when it is used with:

- Datasets comprising hundreds or thousands of variables.

- Data step code that includes a *set* statement that specifies several datasets that comprise many different variables.
- A *merge* statement when it is used with datasets comprising hundreds or thousands of variables.

When used with either the latter two examples the functionality of the *attrib* statement matches some functionality of an 'itemized' *select* clause in a *proc SQL* join.

ADDING FORMAT SPECIFICATIONS

Once you have the variables in the desired order you are ready to move on to other aspects of managing the dataset. One of the variable attributes that can make variable values much easier to comprehend when they are rendered on screen or via reports is the variable's *format*. SAS has an extensive array of formats for both numeric and character variables, but there are quite a few more formats for numeric variables. In addition, SAS provides a procedure (*proc format*) which makes the generation of custom formats a fairly straight-forward task.

A detailed discussion of SAS formats (and informats), is outside the scope of this paper. Detailed explanations and quite a few papers from previous conferences which describe the use of formats are available from SAS' support web site. We can, however, demonstrate how formats are specified in the *attrib* statement.

From the first example *attrib* statement we see that there are 17 virtually identical variables in the sashelp.pricedata dataset and the labels for these variables indicate the values they contain are prices:

```
price1        label = 'Product 1 Unit Price'    length =   8
price2        label = 'Product 2 Unit Price'    length =   8
price3        label = 'Product 3 Unit Price'    length =   8
.
.
price15       label = 'Product 15 Unit Price'   length =   8
price16       label = 'Product 16 Unit Price'   length =   8
price17       label = 'Product 17 Unit Price'   length =   8
```

The dataset does not contain a format specification for any of these variables. Adding an appropriate format specification for each of the 17 variables becomes quite easy with skilled use of the code editor:

```
price1        label = 'Product 1 Unit Price'    length =   8   format = DOLLAR10.2
price2        label = 'Product 2 Unit Price'    length =   8   format = DOLLAR10.2
price3        label = 'Product 3 Unit Price'    length =   8   format = DOLLAR10.2
.
.
price15       label = 'Product 15 Unit Price'   length =   8   format = DOLLAR10.2
price16       label = 'Product 16 Unit Price'   length =   8   format = DOLLAR10.2
price17       label = 'Product 17 Unit Price'   length =   8   format = DOLLAR10.2
```

If variable labels are standardized for the entire dataset and the labels contain key phrases, using regular expressions, editing scripts, or an editor that supports the use of editing macros could easily make this and similar changes to thousands of variables, but only if they are included explicitly in the *attrib* statement. Using a variable list, of course, is another method of accomplishing the same task:

```
attrib
   price1-price17   format = DOLLAR10.2
;
```

This method, however, requires you to add any other variables and attribute changes as needed.

CHANGING VARIABLE LENGTH SPECIFICATIONS

One of the most important aspects of managing larger datasets is appropriate allocation of bytes for each variable. In certain circumstances the default width of character variables is 200 bytes. In all circumstances the default width of numeric variables is 8 bytes. Multiply these defaults by thousands of variables and your talking sizeable chunks of memory.

In almost all instances **you do not want to change the length of numeric variables if the variables contain non-integer values**. If you know for a fact that values in your dataset are, indeed, integers ***and*** you know the range in values of these integers for each of your variables, then you may well be able to save significant amounts of memory.

The following table, along with a very good explanation of how numeric variables utilize memory, is available from SAS' support web site:

Significant Digits and Largest Integer by Length for SAS Variables under Windows			
Length in Bytes	**Largest Integer Represented Exactly**	**Exponential Notation**	**Significant Digits Retained**
3	8,192	2^{13}	3
4	2,097,152	2^{21}	6
5	536,870,912	2^{29}	8
6	137,438,953,472	2^{37}	11
7	35,184,372,088,832	2^{45}	13
8	9,007,199,254,740,992	2^{53}	15

Source: http://support.sas.com/documentation/cdl/en/hostwin/61924/HTML/default/numvar.htm

A good primer for working with numeric variables in SAS can also be found at:

http://support.sas.com/documentation/cdl/en/basess/58133/HTML/default/a001304311.htm

A variable's maximum and minimum values are a couple of the default measures reported by *proc means*:

```
proc means data = sashelp.pricedata ; var _NUMERIC_ ; run ;
```

SAMPLE OUTPUT:

The MEANS Procedure

Variable	Label	Mean	Std Dev	Minimum	Maximum	
region	Region ID	2.1764706	0.7062286	1.0000000	3.0000000	< 8,192: **3**
line	Product Line ID	2.8823529	1.2786011	1.0000000	5.0000000	< 8,192: **3**
product	Product ID	9.0000000	4.9013827	1.0000000	17.0000000	< 8,192: **3**

Once you have generated and pasted the *attrib* statement in the same manner as the previous example, and you know the magnitude of your variable values, you should then be able change the specification for your length attributes fairly easily.

BEFORE EDITING:

```
attrib
    region        label = 'Region ID'              length =   8   format = 6.
    line          label = 'Product Line ID'        length =   8   format = 6.
    product       label = 'Product ID'             length =   8   format = 6.
```

AFTER EDITING:

```
attrib
    region        label = 'Region ID'              length =   3   format = 6.
    line          label = 'Product Line ID'        length =   3   format = 6.
    product       label = 'Product ID'             length =   3   format = 6.
```

Although we changed the length value for just three numeric variables we have saved over 37% of the memory previously allocated by default for those three variables!

CONVERTING VARIABLES FROM ONE TYPE TO THE OTHER

Although variable type conversion cannot be accomplished using the *attrib* statement alone, using it with dataset options for the datasets specified in both the *data* and *set* statements is very efficient.

More thought regarding the management of our sample dataset reveals that even though we have changed the length of the `region`, `line`, and `product` variables, we should instead change the variable type to character: performing mathematic calculations with these variables is not practical as their values are IDs, and we want to make sure others do not try to do so. There are several steps to convert variables and specify correct values for the attributes of the converted variables:

1. Add dataset options to the *set* statement that renames the variables whose types will be converted:

```
set
   sashelp.pricedata (
      rename = (
         region = region_num
         line = line_num
         product = product_num
      )
   )
;
```

(I usually simply append the current variable type to the variable name)

2. Change the attribute specifications appropriately for each of the converted variables:

BEFORE EDITING:

```
attrib
   :
   region        label = 'Region ID'              length =   3   format = 6.
   line          label = 'Product Line ID'        length =   3   format = 6.
   product       label = 'Product ID'             length =   3   format = 6.
   :
```

AFTER EDITING:

```
attrib
   :
   region        label = 'Region ID'              length = $ 6   format = $6.
   line          label = 'Product Line ID'        length = $ 6   format = $6.
   product       label = 'Product ID'             length = $ 6   format = $6.
```

(color utilized to highlight changes)

3. Add to the data step the assignment code necessary to make variable type conversions-making sure to assign converted values to variables as they were originally named:

```
region = put(region_num, 6.0) ;
line = put(line_num, 6.0) ;
product = put(product_num, 6.0) ;
```

4. Add data set options to the *data* statement that drops the old, now renamed, variables:

```
data
   work.pricedata (
      drop = region_num line_num product_num
   )
;
```

We should now have all the data step statements necessary to convert the variable types *and* preserve the original order of the variables:

```
data
   work.pricedata (
      drop = region_num line_num product_num
   )
;
attrib
   .
   .
   region        label = 'Region ID'               length = $ 6   format = $6.
   line          label = 'Product Line ID'         length = $ 6   format = $6.
   product       label = 'Product ID'              length = $ 6   format = $6.
   .
   .
;
set
   sashelp.pricedata (
      rename = (
         region = region_num
         line = line_num
         product = product_num
      )
   )
;
   .
   .
region = put(region_num, 6.0) ;
line = put(line_num, 6.0) ;
product = put(product_num, 6.0) ;
   .
   .
run ;
```

You will find using a full, highly detailed *attrib* statement will help generate notes, warnings, and errors which aid debugging after submitting the next iteration of the data step code; the detailed *attrib* statement provides more information about the intended attributes of the variables for checking by the compiler.

USING THE ATTRIB STATEMENT TO DOCUMENT A DATASET (LIGHTLY)

Hopefully it is now readily apparent the specification of values for most variable attributes for all of the variables in a SAS dataset using an *attrib* statement is quite useful for several data management tasks. In addition to being useful, a patterned layout of the statement can be easily comprehended (if the possible listing of thousands of variables do not bother you) and actually serve as a light form of documentation for the dataset:

```
attrib
   date          label = 'Order Date'              length =   8   format = DATE10.
   sale          label = 'Unit Sale'               length =   8
   price         label = 'Unit Price'              length =   8
   price1        label = 'Product 1 Unit Price'    length =   8   format = DOLLAR10.2
   price2        label = 'Product 2 Unit Price'    length =   8   format = DOLLAR10.2
   .
   .
   price16       label = 'Product 16 Unit Price'   length =   8   format = DOLLAR10.2
   price17       label = 'Product 17 Unit Price'   length =   8   format = DOLLAR10.2
   discount      label = 'Price Discount'          length =   8
   cost          label = 'Unit Cost'               length =   8
   regionName    label = 'Sales Region'            length = $ 7
   productLine   label = 'Name of product line'    length = $ 5
   productName   label = 'Product Name'            length = $ 9
   region        label = 'Region ID'               length = $ 6   format = $6.
   line          label = 'Product Line ID'         length = $ 6   format = $6.
   product       label = 'Product ID'              length = $ 6   format = $6.
;
```

CONCLUSION

Even though there is more coding necessary to use a detailed *attrib* statement for variable attributes, the statement itself can be subsequently used to help with several dataset management tasks: the ordering of variables, adding or changing variable labels, adding or changing format and informat specifications, changing the allocation of memory for variables, and converting variables of one type to the other.

Using macro code to generate the *attrib* statement for a dataset generated by an early version of data step code and then inserting the statement into the next version of the same data step code can be quite easy. Given this ease, regenerating and replacing the *attrib* statement for each new iteration of the code can become routine and help debug the data step code itself: The only subsequent editing of the statement required would be for any new variables introduced by the latest iteration of the data step code.

REFERENCES

- First, Steven (2009), "Understanding the SAS DATA Step and the Program Data Vector," *Proceedings of SAS Global Forum 2009*
  http://support.sas.com/resources/papers/proceedings09/136-2009.pdf
- Philp, Stephen (2006), "Programming with the KEEP, RENAME, and DROP Data Set Options," *Proceedings of the Thirty-First Annual SAS Users Group International Conference*
  http://www2.sas.com/proceedings/sugi31/248-31.pdf
- Schreier, Howard (2005), "Let Your Data Power Your DATA Step: Making Effective Use of the SET, MERGE, UPDATE and MODIFY Statements," *Proceedings of the Thirtieth Annual SAS Users Group International Conference*
  http://www2.sas.com/proceedings/sugi30/251-30.pdf
- Whitlock, Ian (2006), "How to Think Through the SAS DATA Step," *Proceedings of the Thirty-First Annual SAS Users Group International Conference*
  http://www2.sas.com/proceedings/sugi31/246-31.pdf

ACKNOWLEDGMENTS

I would like to thank the participants in the Michigan SAS Users group for their encouragement and suggestions.

ABOUT THE AUTHOR

Phil Wright graduated from the University of Michigan in 1986 with a Bachelors degree in Psychology. He first sat down in front of a PC when his first research project purchased their first PC and asked him to learn their word processing application (FinalWord) and then teach it to the rest of the staff. He has been in front of a PC ever since. Phil has been using SAS for over 15 years; specializing in the conversion of legacy data files, data management, reporting, *proc SQL,* ODS, and Macro programming.

CONTACT INFORMATION

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

Name:	Philip A. Wright
Enterprise:	Inter-university Consortium for Political and Social Research (ICPSR), Institute for Social Research (ISR), University of Michigan
Address:	P.O. Box 1248
City, State ZIP:	Ann Arbor, Michigan 48106-1248
Work Phone:	734-615-7886
Fax:	734-647-8200
E-mail:	pawright@umich.edu
Web:	http://www.icpsr.umich.edu