Paper 069-2010

Using the Descriptor Portion of a SAS® Data File

Dragos Daniel Capan, Canadian Institute for Health Information, Toronto, Ontario

ABSTRACT

The descriptor portion is often an overlooked component of a SAS® data file. The purpose of the paper is to explain what the descriptor portion is, when it is created, and how it can be accessed. Four examples will illustrate the usefulness of the descriptor. To give context and make the first two examples clear, the paper will briefly remind the reader how SAS processes a DATA step. The first two examples will demonstrate how using the information in the descriptor portion can make the SAS code more efficient. The last two examples will show how using the descriptor combined with the power of macro processing can make the programmer's life much easier and the SAS code more elegant.

INTRODUCTION

Apart from the actual data a SAS data file contains information on the physical data set and on the individual variables into what is called the descriptor portion. Thus, this information consists in the number of observations the data set has, the date the data set was created and last modified, the number of indexes, the number of pages and the page size, the variable names and their type, length, format, informat, label etc.. This information can be accessed and hence used to different purposes as a few examples in this paper demonstrate.

THE DESCRIPTOR PORTION OF A SAS DATA SET

Figure1. below depicts the two parts of a SAS data set: the descriptor and the actual data.

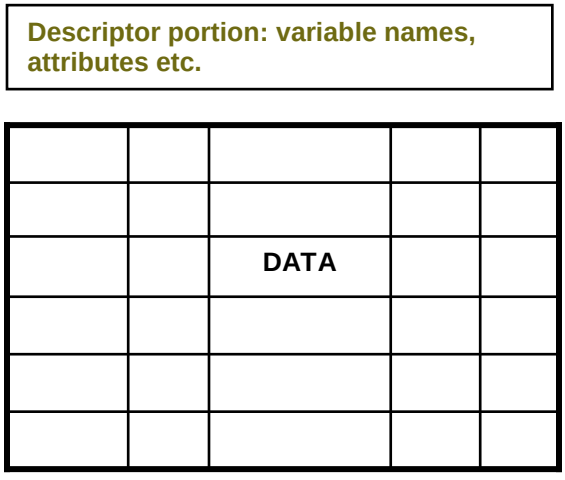

Figure 1: Data set

To view the information in the descriptor portion of a SAS data set, one can use either a PROC CONTENTS or PROC DATASETS with the CONTENTS statement as exemplified below:

```
proc contents data='libref'.dataset; run;
```

OR

```
proc datasets library='libref' nolist;
```

```
contents data=dataset;
quit;
```

To see how and when the descriptor portion is created and to better understand the first 2 examples, the next section briefly reminds the reader how SAS processes a DATA STEP.

PROCESSING A SAS DATA STEP

A SAS DATA step is processed in two phases: compilation & execution. During the **compilation phas**e each statement is scanned for syntax errors. *When the compilation phase is complete, the descriptor portion of the new data set is created.* If the DATA step compiles successfully, then the **execution phase** begins. During the execution phase, the DATA step reads and processes the input data.

Let's imagine that we want data set B to be a copy of data set A to which we also want to add another variable X calculated using the existing variable Y from A. The SAS code would look like this:

```
data B;
  set A;
  x=y+3;
run;
```

At the end of the compilation phase of the above code, the descriptor portion of the data set B will be created and it will be similar but not the same to the descriptor of the data set A. The next diagram shows what happens during the next phase, the execution:

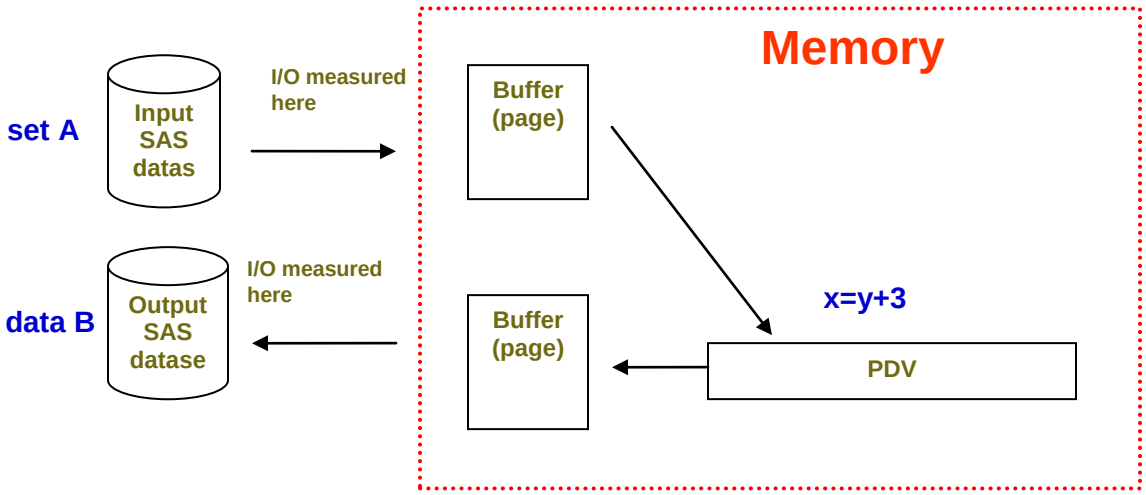

Figure 2: Data step processing

The SET statement reads data from its original location by loading one page at a time into the buffer. Loading a page constitutes one I/O operation. The more pages SAS reads the more I/O operations. Once a page is loaded, each record/observation in the page is loaded into the program data vector (PDV) where it is processed. Next, the record is sent to another buffer where once the number of records that make a page are loaded they are sent to the output data set (the DATA statement). This last part is the 'writing' and again, each loading of a page from the buffer to the output data set makes a I/O operation. At the creation of the data set one can increase the page size which will decrease the number of I/O at the expense of increased memory usage.

EXAMPLES OF USING THE DESCRIPTOR PORTION

EXAMPLE 1

Upon running a SAS code you find out that your new data set name is misspelled and that some of the variable names are not the same as the ones you used when writing a macro that uses this data set. So, you need to modify the name of the data set and the variable names. For consistency you also need to format one of them.

Let's say we need to rename the data set DSD to DAD and the variable names *dsd_trans* to *dad_trans* and *admission_date* to *admdate* and format the latter. First, we will look at the least desirable way of solving this problem:

```
data mylib.DAD;
set mylib.DSD;
  rename dsd_trans=dad_trans
       admission_date = admdate;
  format admdate date9.;
run;
```

Why is this solution not recommended ? Well, in order to change some names and apply a format, the solution above reads and writes all data without modifying anything to it. The renaming and formatting take place in the compilation phase and therefore reading and writing data again is unnecessary. The best way of accomplishing the task is using PROC DATASETS as described below:

```
proc datasets lib=mylib;
  change DSD=DAD;
  modify DAD;
  rename dsd_trans = dad_trans
        admission_date=admdate;
  format admdate date9.;
quit;
run;
```

In this scenario, the DATASETS procedure accesses the descriptor portion of the DSD data set and makes the desired changes.

To see the gain in efficiency I ran this example on a powerful server. The data set DSD has around 2.5 million observations and 50 variables. The data step had a real time of 30.64 seconds and a CPU of 12.07 seconds. PROC DATASETS used 0.01 real and CPU time.

EXAMPLE 2

We need to create a macro variable that holds the number of observations in a certain data set.

As we all know, there are multiple ways of completing a task in SAS. I will present 3 ways to create the macro variable that holds the number of observations in a SAS data set: starting with the least efficient to the most efficient method.

The code below is inefficient because to create the macro variable 'nr_obs' we only use the last row . However, the code reads and writes all records.

```
data mylib.dad;
 set mylib.dad end=last;
 if last then call symput('nr_obs',_N_);
run;
```

The real time for this SAS code was 28.26 sec while the CPU time was 13.19 sec.

If we eliminated the writing part by using the DATA _NULL_ statement we get a considerable reduction in both real time and CPU time (4.18 and 3.44) :

```
data _null_;
 set mylib.dad end=last;
 if last then call symput('nr_obs',_N_);
run;
```

While the second method works a lot better it is still not perfect because it has to read all the records to be able to reach the last one and create the macro variable.

This is when we need to remember that the descriptor portion holds the number of observations in a data set. A quick to solve our problem way would be to run a PROC CONTENTS and manually create the macro variable by ways of a

LET statement. However, there is a better way of achieving this: the SET statement has an NOBS = *name* option that creates an automatic variable which holds the number of observations in that data set. What makes this method better is that this information is read at compilation time and hence there is no need to actually execute the SET statement:

```
data _null_;
 if 0 then set mylib.dad nobs=nr;
 call symput('nr_obs',nr);
 stop;
run;
```

The condition "If 0" will always return false and hence the SET statement is never executed. However, the variable *nr* is initialized with the number of observations in the data set DAD during the compilation phase. Therefore, no reading and no writing is being done. The STOP statement is to avoid a note in the log about continuous looping. Since the SET statement is not executed, there is no END OF FILE and hence there is no instruction to end the DATA STEP. The real time and cpu time are now 0.00 seconds. This method doesn't work with an edited data set (through the MODIFY statement) or with data step or SQL views (Jack Hamilton,)

The first 2 examples were about how to use the descriptor information to solve simple tasks in a more efficient way. The next 2 examples are more about how this information can make your life easier by not only helping you solve problems but also doing so in an elegant way.

EXAMPLE 3

Create a macro variable that holds all the variable names in a data set (or only the numeric/character ones).

In my SAS programming experience there were times when I found it useful to have all or some of the variable names from a data set into a macro variable. I found this especially helpful when using procedures like PROC SCORE where you need to specify the variables needed to score the data or PROC LOGISTIC where you have to specify the list of covariates. When using PROC SCORE, the variables are found in the scored data set (SCORE = *data set*) which can be for example the data set that holds the coefficients from a logistic regression.

We first run a PROC CONTENTS on the data set that contains the coefficients and use the OUT = option to create a data set that holds the information in the descriptor. At the same time we keep only the variable that holds the variable names, called *name* and also delete some of its values (a few variable names automatically created by SAS when running the logistic regression procedure).

```
proc contents data=parameters out=var_names (keep = name where=(name not in
  ('Intercept', '_LINK_', '_LNLIKE_', '_NAME_', '_STATUS_', '_TYPE_'))) noprint
run;
```

To create the macro variable we can use PROC SQL:

```
proc sql noprint;
select name into: covariates separated by " "
    from var_names;
quit;
```

Now, the macro variable is created and ready to be used in PROC SCORE:

```
proc score data=hsmr_data score=parameters out=scored_data TYPE=PARMS;
 var &covariates;
run;
```

This method is not only elegant (who wants to type 70 variable names ?) but also reduces the possibility of an error whether the error being a typo or omitting a variable from the list.

EXAMPLE 4

We have a data set with 120 numeric and 10 character variables. We need to replace the value of -2009 from the numeric variables with the dagger character. Obviously, we will need to transform all the numeric variables into character and in doing so, keep the format and their order in the final data set the same.

We start as before by creating a data set that holds the variable names, their type, format and physical location in the data set.

```
proc contents data=epub noprint
  out=variables (keep = name type varnum formatl formatd);
run;
```

Since the format is made of 2 variables which specify the format length and decimal (formatl and formatd), we put the two together and also restrict the new data set to keep only the numeric variables:

```
data variables_num;
set variables (where=(type=1));
 length varformat $  7;
 if formatl=0 then varformat='best12.'; else
 varformat=put(formatl,1.)||"."||put(formatd,1.);
run;
```

Next, we will create macro variables that will help us accomplish the task of transforming all the numeric variables into character variables:

- *for1, for2, for3* … will hold the FORMAT of each variable that needs to be transformed
- *mac1, mac2, mac3* … will hold the NAME of each variable
- *ch1, ch2, ch3* … will be the temporary names for the variables

```
data _null_;
set variables_num end=last;
 call symput('for'||left(put(_N_,4.)),varformat);
 call symput('mac'||left(put(_N_,4.)),name);
 call symput('ch'||left(put(_N_,4.)), 'ch'||left(put(_N_,4.)));
 if last then call symput('end',put(_N_,8.));
run;
```

So macro variable *mac1* will resolve to the name of first numeric variable, *for1* will resolve to its format and *ch1* will resolve to ch1 which will be its temporary name while a character variable. We also create a macro variable called 'end' to hold the number of numeric variables we need to transform in the next step:

```
data epub_ch (drop= %do p=1 %to &end;&&mac&p %end;
           rename=(%do p=1 %to &end; &&ch&p=&&mac&p  %end;));
set epub ;

%do p=1 %to &end;
  &&ch&p=strip(put(&&mac&p,&&for&p));
  if &&ch&p in: ("-2009") then &&ch&p ="†";
%end;

run;
```

What is left to do is to ensure the order is the same as in the initial data set. We will use again the outputted data set from PROC CONTENTS which we called 'variables' to first order our variable names by 'varnum'. Varnum is the variable that holds the sequence in which the variables in the initial data set are stored. Then we use a PROC SQL to create a macro variable to holds these variable names in their initial order.

```
proc sort data=variables; by varnum; run;

proc sql noprint;
 select name into :inorder separated by ', ' from variables;
quit;
```

Another PROC SQL below creates another data set with the variables in order this time.

```
proc sql noprint;
 create table epub_table as
 select &inorder from epub_ch; quit;
```

CONCLUSION

The information in the descriptor portion of a SAS file can be very useful. It can help you solve apparently difficult tasks, write efficient, elegant, error proof SAS code. The examples in this paper illustrated only a few of the possible uses of this information.

REFERENCES

SAS® 9.1.3 online documentation
SAS® Certification Prep Guide: Advanced Programming for SAS9 Second Edition
Hamilton, Jack 2001. "How many observations are in my dataset" *Proceedings of the Twenty sixth Annual SAS® Users Group International Conference,* Long Beach, CA, paper 95-26.

CONTACT INFORMATION

Dragos Daniel Capan
Canadian Institute for Health Information
4110 Yonge st, Toronto, ON
Phone: 1416 5495559
Email: dcapan@cihi.ca