

Overcome Programming Problems with SAS® Automatic Variable N

Louie Huang, Baxter BioScience, Westlake Village, California 91362, USA

ABSTRACT

The automatic variable `_n_` is automatically created by the DATA step or DATA step statements. It exists in the program data vector but is not exported to the data set being created. The value is retained from one iteration of the DATA step to another. The power of this variable in programming is often overlooked. This presentation is intended to demonstrate how to use the automatic variable `_n_` to overcome complicated programming problems. A few practical examples will be introduced in this presentation.

INTRODUCTION

What is the automatic variable `_n_`? When you open a SAS data file in a data library, you will notice the observations are numbered at the first column. The automatic variable `_n_` represents the observation numbers. This variable hidden in the DATA step is simple yet powerful in programming practice. This paper illustrates how the automatic variable `_n_` is used in real world examples. However, the variable names and data used in this paper were modified for confidentiality reasons.

Example 1: Overcome ODS template bugs

Let's suppose that your clients would like to have a PDF or RTF document without showing row lines (Table 1). You think you can make your clients happy by simply setting the frame and rules attributes to "frame=hsides" and "rules=groups" in the ODS template, respectively. However, as you compare Table 2 with Table 1, you will notice that Table 2 lacks a table header line, Subject ID "12345c", and product "ABCDEFG2". The relevant bugs occurring in the SAS versions 8.2 and 9.1 are listed in Table 3 for your information.

Table 1. Output Example Expected to Create (the second page)

Subject ID	Product	System Organ Class
12345c	ABCDEFG2	Musculoskeletal and Connective Tissue Disorders Metabolism and Nutrition Disorders
	ABCDEFG1	Metabolism and Nutrition Disorders
12345d	ABCDEFG2	Gastrointestinal Disorders
12345e	ABCDEFG1	General Disorders and Administration site Conditions

Table 2. Output Created by setting "frame=hsides" and "rules=groups" (the second page)

Subject ID	Product	System Organ Class
		Musculoskeletal and Connective Tissue Disorders Metabolism and Nutrition Disorders
	ABCDEFG1	Metabolism and Nutrition Disorders
12345d	ABCDEFG2	Gastrointestinal Disorders
12345e	ABCDEFG1	General Disorders and Administration site Conditions

Table 3. Description of Bugs Occurring in PDF and RTF Output for SAS® Versions 8.2 and 9.1

Description of Bugs	Bugs Occurred in	
	Output format	SAS Version
1. No group line from the second to last page	PDF	SAS 8.2
2. When contents of the same group variables carry to the following page, the group variables do not display.	PDF, RTF	SAS 8.2, 9.1
3. No bottom line from the 1 st to second last page	RTF	SAS 8.2, 9.1

You can overcome the problems listed in Table 3 by the following three steps.

1) Create a variable *pgbrk* at the DATA step by taking advantage of the automatic variable *_n_*. This variable functions to control the number of pages, and contents of each page.

```
data mydata1;
  set wuss13.mydata1; /* one page can hold 19 lines of output contents in this case */
  if _n_ le 19 then pgbrk=1; /* page 1 */
  else if 20 le _n_ le 38 then pgbrk=2; /* page 2 */
  else if 39 le _n_ le 57 then pgbrk=3; /* page 3 */
  else if 58 le _n_ le 76 then pgbrk=4; /* page 4 */
  else if 77 le _n_ le 95 then pgbrk=5; /* page 5 */
  else pgbrk=6; /* page 6 */
run;
```

You should be aware that the number of lines of output in each page is affected by a lot of things, such as text of table headers, footnotes, titles, and page margins. If you are not sure what the maximum number of lines is for each page after revising your SAS file, you simply count the rows of the first page in your new output. Then you use this information to reset the values of the variable *pgbrk*. In addition, I would like to emphasize that the values of the variable *pgbrk* can be conveniently adjusted to suit your needs.

2) Convert the values in variable *pgbrk* to a macro variable by PROC SQL. The macro variable *brklist* contains a list of the unique page number values.

```
proc sql noprint;
  select distinct pgbrk
    into :brklist separated by '#'
  from mydata1;
quit;
```

3) Place your procedure (i.e., PROC REPORT, PROC TABULATE) within a macro with a %DO LOOP function. In this particular example, one page is generated as the PROC REPORT procedure repeats one time.

```
ods pdf file='C:\wuss13\v8_soc2.pdf' style=pdf_style;
ods rtf file='C:\wuss13\v8_soc2.rtf' style=rtf_style;
%macro repit;
  %do i=1 %to &sqlobs;
    %let brk=%scan(&brklist,&i,#);
    ods proclabel "SOC,page &i";
  proc report data=mydata1 split='*' nowd;
    where pgbrk=&brk;
    columns subject product meddra ser sev rel;
```

```

define subject / group 'Subject ID' style=[just=L cellwidth=120];
define product /group 'Product' style=[just=L cellwidth=160];
define meddra / 'System Organ Class' style=[just=L cellwidth=640] ;
define ser / 'Serious?' format=$seryn. style=[just=L cellwidth=100];
define sev / 'Sevurity' style=[just=L cellwidth=145];
define rel / 'Relationship' style=[just=L cellwidth=145];
title 'List of System Organ Class Grouped by Subject and Product';
run;
%end;
%mend;
%repit
ods _all_ close;

```

The shortcoming of using the method introduced above to overcome ODS template bugs is that you need to summarize your data by another procedure, i.e., PROC MEANS, or PROC SQL.

Example 2: Use Automatic variable `_n_` and array function to update small data files

For instance, if you want to create a new data file *newdata* from the old data file *olddata*, since you have to keep some variables from the old file. You can significantly reduce your coding by using the automatic variable `_n_` in an array statement.

```

data newdata;
array ar1(1:11) _temporary_ (310 310 370 410 250 380 330 370 310 380 290);
array ar2(1:11) _temporary_ (270 260 300 390 210 350 365 385 400 410 320);
set olddata(keep=Sequence Patient); /* pick variables from the old file */
Period1=ar1(_n_); /* the new variables needed to create */
Period2=ar2(_n_);
run;

```

Of course, alternative ways are always available. You can perform the same task by using the INSERT and VALUES clauses of SQL or by reading the instream data through the DATA step. However, the way introduced by this example is the most efficient method.

Example 3: Use automatic variable `_n_` by the DATA step to create macro variables

When you want to create macro variables by the DATA step, it is not necessary to create a counter variable and use the RETAIN function if the automatic variable `_n_` is introduced, thus making the coding shorter and more intuitive.

Coding with the automatic variable `_n_`

```

data _null_;
set mydata end=eof;
if eof then call symput('num_inv',_n_);
call symput(compress("inv"||_n_),surgeon);
run;

```

Coding without the automatic variable `_n_`

```

data _null_;
set surge_ID end=eof;
retain Counter 1;
if eof then call symput('Num_inv',Counter);
call symput(compress("inv"||counter),surgeon);
Counter+1;
run;

```

List of the macro variable examples

Macro	Value
&inv_num	16
&inv1	Davidson
&inv2	Smith III
&inv3	Johnson,Jr.

Example 4: Use automatic variable `_n_` in SAS/GRAPH®

If you want to generate a graph as shown in Figure 1, in which the recovery rate displays in a descending order and the subject identifications must be masked for confidentiality reasons. There are several methods to accomplish the job. No matter what methods are used, it is necessary to create a sorted data file by the PROC SORT procedure.

```
proc sort data=wuss13.Revdata out=sorted;
  by descending rev_rate;
run;
```

Method 1:

You probably prefer to use the IF ... THEN statement. It definitely does the job. Imagine that 30 IF ... THEN statements have to be used. That is a lot of boring coding!

Method 2:

Use CASE clause by SQL, by which the code can be shortened a little. It is still pretty lengthy and boring.

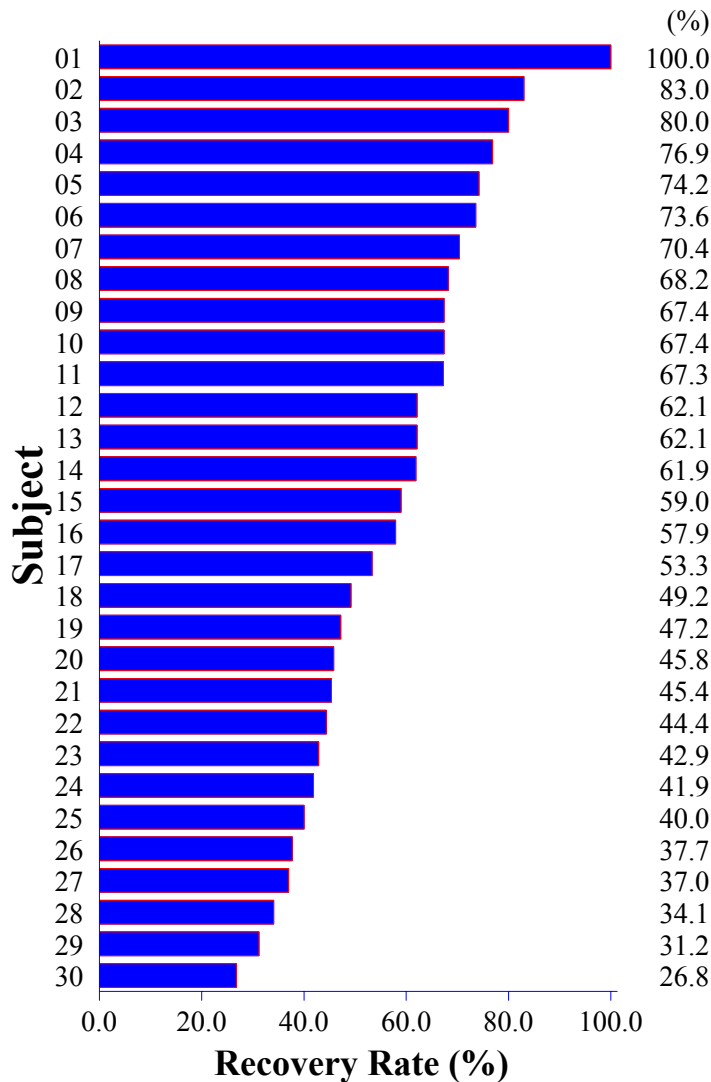
```
proc sql;
create table final as
select case subjid
  when 's1043' then '01' when 's1021' then '02'
  when 's2028' then '03' when 's2016' then '04'
  when 's1017' then '05' when 's1022' then '06'
  when 's1020' then '07' when 's2026' then '08'
  when 's1015' then '09' when 's1048' then '10'
  when 's1019' then '11' when 's1008' then '12'
  when 's1035' then '13' when 's2010' then '14'
  when 's2033' then '15' when 's1093' then '16'
  when 's1503' then '17' when 's2014' then '18'
  when 's0533' then '19' when 's1063' then '20'
  when 's2001' then '21' when 's1032' then '22'
  when 's1067' then '23' when 's2011' then '24'
  when 's2012' then '25' when 's2015' then '26'
  when 's2041' then '27' when 's1024' then '28'
  when 's2038' then '29' else '30'
end as subjid2, rev_rate
from sorted;
quit;
```

Method 3:

You can use the automatic variable `_n_` in this kind of programming situation. It performs the same job function with only a couple of lines of programming code.

```
data final(keep=subjid2 rev_rate);  
  set sorted;  
  subjid2=put(_n_,z2.); /* automatic variable _n_ was formatted to z2. format */  
run;
```

Figure 1. Graph Display for Example 4



Example 5: Solve complicated SAS/GRAPH issues using the automatic variable `_n_`

For example, you need to create plots of blood sugar level over treatment days as shown in Figure 2 for over 100 patients. Three different treatment days are displayed by concatenating the numbers together. The technique issue is that the annotated text has to be displayed in different

angles and positions, otherwise some of the annotated text would overlap. You can break down your data file into sub data files by subject, each observation is assigned to a unique number by the automatic variable `_n_`, then the sub data files are appended into one big file. The programming code is shown as follows:

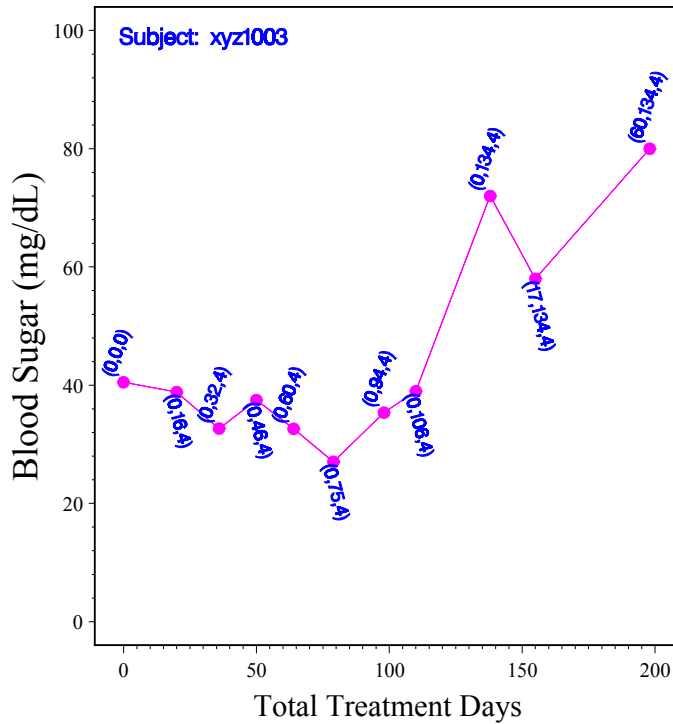
```
proc sql noprint;
  select count(distinct subjid)
  into :ct /* macro variable ct is the total number of subjects */
  from mydata;
  select distinct subjid
  into :subj1 - :subj%left(&ct) /* one macro variable for each subject */
  from mydata
  order by subjid;
quit;

%macro add_num;
  %do i=1 %to &ct;
    data sub&i;
    set mydata;
    where pt=&&subj&i;
    num=_n_ /* the variable num is inserted into each dataset by subject */
    %if &i=1 %then %do;
data mydata2;
  set sub&i;
  %end;
  %else %do;
proc append base=mydata2 data=sub&i;
  %end;
run;
  %end;
%mend;
%add_num
```

The numeric variable `num` can make the job much easier when creating the annotation datasets. One annotated data file will be generated for each patient by placing the following programming code under a macro with the `%DO` function.

```
data anot(drop=bld_sugar treat_day anotat num);
  length color function style $8 text $40;
  retain function 'label' xsys ysys '2' hsys '1' style 'swiss' when 'a';
  if _n_=1 then do;
    y=100; x=30;
    text="Subject: xyz1003";
    color='blue'; size=3.5;
    output;
  end;
set mydata2;
  y=bld_sugar; x=treat_day;
  text=trim(anotat);
  color='blue'; size=2.8;
  if mod(num,2)=1 then do;
    position='3';
    angle=70;
  end;
  else do;
    position='6';
    angle=-75;
  end;
  output;
run;
```

Figure 2. Graph Display for Example 5



CONCLUSION

SAS[®] provides many powerful tools to manipulate data and create customized reports. Using the automatic variable `_n_` is only one of the many examples. This paper only demonstrated five examples for your reference to reduce or dramatically simplify your coding, flexibly manage your data and effectively solve your practical problems. However, there are more specific examples of using automatic variable `_n_` to facilitate your programming tasks.

ACKNOWLEDGEMENTS

The author would like to express his special thanks to Bill Knowlton for his support and valuable suggestions in reviewing this paper.

CONTACT INFORMATION

You can send your comments, questions, and inquiries to:

Louie Huang, Senior Statistical Analyst
Baxter BioScience, Baxter Healthcare Corporation
One Baxter Way
Westlake Village, CA 91362
Tel: 805-372-3487
Fax: 805-372-3462
Email: louie_huang@baxter.com

TRADEMARK INFORMATION

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.