

Tips about Using trailing @

Jingren Shi, Syntel Inc., Troy, Michigan
Shiling Zhang, Wayne State University, Detroit, Michigan

Abstract

This paper presents various tricks and simple efficient techniques that can benefit all kinds of SAS® users. An inexperienced SAS programmer often has a question about how to read a text file into a SAS data set in desired form. Others may have difficulties when a raw text file is in an uncommon type. This paper will discuss about using the trailing @, @@ and # to make one's codes more efficiency. All examples used here are based on questions posted on comp.soft-sys.sas or sas-l

Terminology and Definition

@ --- Line-hold specifiers^[1] Line-hold specifiers keeps the pointer on the current input line when

- a data line is read by more than one **INPUT** statement (trailing @).
- one input line has values for more than one observation (double trailing @@).

Normally, each **INPUT** statement in a data step reads a new data line into the input buffer. When one uses a trailing @, the following occurs:

- The pointer position remains the same.
- No new record is read into the input buffer.
- The next **INPUT** statement for the same iteration of the data step continues reading the same record.

A line held with a trailing @ is released automatically when

- A null **INPUT** statement executes:
input;
- The system retains to the top of the data step to begin the next iteration.

A double trailing @@ holds an input line for further iterations of the data step. The SAS system releases a line held by A double trailing @@

- immediately if the pointer moves the end of the line
- immediately a null **INPUT** statement executes:
input;

- when the next iteration of the data step begins if an **INPUT** statement with a single trailing @ executes later in the data step:
input @;

--- Line pointer controls^[2] Line pointer controls specify an input line from which an **INPUT** statement is to read a value.

Input buffer^[3] --- Area of memory into which each record of raw data is read when an **INPUT** statement executes.

Program data vector(PDV)^[4] --- Area of memory where the SAS system builds your data set, one observation at a time. When the program executes, data values are read from the input buffer or created by SAS language statements and assigned to the appropriate variables in the program data vector. From here the variables are written to the SAS data set as a single observation.

Case 1

One wants to read daily temperature data into a SAS-File. Currently the data are formatted

```
Day Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1 3 4 3 5 7.3 9.2 5.5 6 8 6.5 4.8 3.2
2 3.5 4.4 5 5.3 3.2 5.9 7 6.3 7.4 5.7 10 3.2
3 6.3 7.4 5.3 3.2 5.9 7 5.7 10 3 5 7.3 1.9
....
```

One would prefer to use data of the format

```
Date      temperature
1/1/97     3
2/1/97     4
3/1/97     3
...
```

An inexperienced SAS programmer may consider using a data step and a 'PROC TRANSPOSE' to get the desired data set form. This approach is inefficient.

This is a typical example of use a trailing @. One can use two trailing @s to hold a line position, and read the data into a desired form.

```
data temp(keep =temp date);
```

```

input day @;      ❶
do m = 1 to 12;
  input temp @;    ❷
  date = put(m,z2.)||'/'||put(day,z2.)||'/'||'97';
  output;
end;
cards;
1 3 4 3 5 7.3 9.2 5.5 6 8 6.5 4.8 3.2
2 3.5 4.4 5 5.3 3.2 5.9 7 6.3 7.4 5.7 10 3.2
3 6.3 7.4 5.3 3.2 5.9 7 5.7 10 3 5 7.3 1.9
....
;

```

Case 2

One has a data file that has household records and personal records. The household record and the personal record have different file layouts. The question is how to create a household data set and a personal data set. The file has the following simplified structure:

```

HH 1 2
PP 1 2 5
HH 2 3
PP 1 3 7
PP 1 4 5

```

Where HH is a household indicator and PP is a personal indicator.

In this example, one can read the first field first. If the value indicates that it is a household record, then one can use the household record layout to read in the rest of fields. Otherwise, it is a personal record. With this understanding, the codes should be simple.

```

data hh(keep=indicate a b) pp(keep=indicate d e f);
input indicate $2. @ ;
if indicate='HH' then do;
  input a b ;
  output hh;
end;
else do;
  input d e f;
  output pp;
end;
cards;
HH 1 2
PP 1 2 5
HH 2 3
PP 1 3 7
PP 1 4 5
;

```

Case 3

One has a survey data file. The questionnaire divides into several sections. Within each section, there are several questions. The file has the

following simplified structure. For each record, there are three sections with different numbers of questions.

```

sec_id1 q1 q2 q3 q4
sec_id2 q1 q2
sec_id3 q1 q2 q3 q4 q5

```

where q1 – q5 are questions under the section.

This request is similar as case 2. Hence, one can use the above approach. However, one needs to output a record after all three lines are read in.

```

data srv_ds(drop=sec_id);
length sec1_q1 - sec1_q4 $8
      sec2_q1 - sec2_q2 $5
      sec3_q1 - sec3_q5 $4;
retain sec1_q1 - sec1_q4
      sec2_q1 - sec2_q2
      sec3_q1 - sec3_q5;

input sec_id $7. @ ;
if sec_id='sec_id1' then input sec1_q1 - sec1_q4;
else if sec_id='sec_id2' then input sec2_q1-
sec2_q2;
else do;
  input sec3_q1- sec3_q5;
  output;
end;
cards;
sec_id1 q1 q2 q3 q4
sec_id2 q1 q2
sec_id3 q1 q2 q3 q4 q5
;

```

Since every three lines consist of one record, using the line pointer control # is straightforward. Here are codes.

```

data srv_ds;
length sec1_q1 - sec1_q4 $8
      sec2_q1 - sec2_q2 $5
      sec3_q1 - sec3_q5 $4;
input #1 @9 sec1_q1 - sec1_q4
      #2 @9 sec2_q1 - sec2_q2
      #3 @9 sec3_q1 - sec3_q5;
cards;
sec_id1 q1 q2 q3 q4
sec_id2 q1 q2
sec_id3 q1 q2 q3 q4 q5
;

```

Case 4

One has a lab testing data file. The ASCII file layout looks like as follows,

```
test1 11 2
```

```
test2 1 2 5
test3 2
```

where the first field is the test subject and the rest of the fields are repeated testing results for the subject. One wants the final file has following format,

```
test_id count result
```

```
test1 1 11
test1 2 2
test2 1 1
test2 2 2
test2 3 5
test3 1 2
```

The difficult part of this example is that the numbers of tests vary from subjects to subjects. However, if one reads each line with a trailing @, one needs not to know the numbers of tests. The logic is straightforward. First, one reads the first field, and then reads in rest of fields and outputs it. Second, one needs to check when a line is end. If it is end, then one begin to read the next line.

```
data t1;
  length test_id $8 cnt result 8;
  infile cards eof=eof ;
  input @1 test_id $5. @ ;      ❶
  do while (1);
    cnt+1;
    input result ?? @;          ❷
    if result = '.' then do;    ❸
      cnt=0;
      input @@;                 ❹
      leave;
    end;
    else output;
  end;
  return;
eof: stop;
cards;
test1 11 2
test2 1 2 5
test3 2
;
```

❶, ❷ hold line position to prevent SAS from going to a new line.

❸ tests if it is the beginning of a new line.

❹ holds the current line position.

It is much simpler if one use double trailing @@ and a retain statement in this example.

```
data t1(drop=dummy);
  length test_id dummy $8;
  retain test_id;
  input dummy @@ ;
  if input(dummy, ?? best.) ne . then do;
```

```
    result = input(dummy, 8.);
  output;
end;
else test_id=dummy;
cards;
test1 11 2
test2 1 2 5
test3 2
;
```

Case 5

One has a text file in which each row represents a variable and its value. The file looks like as follows,

name	mike	jeff	carl	bill	sara	judy
age	24	33	19	28	24	22
gender	m	m	m	m	f	f
share	15%	18%	87%	44%	20%	5%
income	\$45	\$33	\$23	\$34	\$67	\$18

One wants the output as follows,

NAME	AGE	GENDER	SHARE	INCOME
mike	24	m	0.15	45
jeff	33	m	0.18	33
carl	19	m	0.87	23
bill	28	m	0.44	34
sara	24	f	0.20	67
judy	22	f	0.05	18

An inexperienced SAS programmer will often read all data as character values. Then he will use 'PROC TRANSPOSE' to get the desired data set form and another data step to change some character variables into numeric variables.

Since the field is transposed, each field has a fixed length, hence it is easy to write a formula for a column control pointer. One could use that point together with a trailing @ and a line pointer control # to read the text file into a SAS data set.

```
data t1(drop=i j);
  do i = 1 to 6;              ❶
    j=(i-1)*6+8;             ❷
    input #1 @j name $
          #2 @j age
          #3 @j gender $
          #4 @j share PERCENT6.
          #5 @j income DOLLAR5. @;  ❸
    output;
  end;
  list;
cards;
name mike jeff carl bill sara judy
age 24 33 19 28 24 22
gender m m m m f f
share 15% 18% 87% 44% 20% 5%
income $45 $33 $23 $34 $67 $18
;
```

❶ Total six observations will be in a final data set.

- ② It forms a formula for a column control pointer.
- ③ The trailing @ holds line position to prevent SAS from going to a new line so that the next **INPUT** statement for the same iteration of the data step continues reading the same record.

Case 6

One has a data file that has one record per subject. The file has the following simplified structure:

psu	subj_id	cesd1	cesd2	cesd3
0	5	0	2	3
0	6	1	2	2
1	9	1	3	4
1	3	7	3	6
1	4	6	4	7
1	7	3	1	1
2	2	2	0	0

One wants the final data set sorted by psu. The new_id is numbered within psu, and time is numbered within new_id. This is a repeated (three times) measurements of cesd per subject, which is nested with psu. One wants to make the final data set look like the following:

psu	new_id	time	cesd
0	1	1	0
0	1	2	2
0	1	3	3
0	2	1	1
0	2	2	2
0	2	3	2
1	1	1	1
1	1	2	3
1	1	3	4
1	2	1	7
1	2	2	3
1	2	3	6
1	3	1	6
1	3	2	4
1	3	3	7
1	4	1	3
1	4	2	1
1	4	3	1
2	1	1	2
2	1	2	0
2	1	3	0

The problem looks difficult. It is the same as case 1 except the request of creating two new variables **NEW_ID** and **TIME**. With this understanding and by using @ codes should be short and simple. Assume that records with the same psu code are grouped together. All can be done within a single data step.

```
data t1;
```

```
retain flag;
input psu subj_id @;
if _n_=1 then flag=psu;
if psu ne flag then do;
    new_id=0;
    flag=psu;
end;
new_id+1;
do i = 1 to 3;
    time=i;
    input cesd @;
    output;
end;
drop i subj_id flag;
cards;
0      5      0      2      3
0      6      1      2      2
1      9      1      3      4
1      3      7      3      6
1      4      6      4      7
1      7      3      1      1
2      2      2      0      0
;
```

- ① The function of variable **FLAG** is to check when the value of **PSU** changes. The value of variable **FLAG** needs retain to next iteration.
- ② It holds line position to prevent SAS from going to a new line until last field has been reach.
- ③ When the value of **PSU** changes, the value of variable **NEW_ID** needs reset to zero and the value of variable **FLAG** needs reset to the current value of **PSU** for next iteration.
- ④ It creates variable **TIME**, **CEDSD** and outputs them three times. The trailing @ holds line position to prevent SAS from going to a new line until last field has been reach.

Case 7

One has a flat file with a comma delimiter and each record expands to two lines. There is also a header line that one wants to omit. An example is below:

```
hd1,hd2,hd3,hd4,hd5,hd6,hd7,hd8
87-1234-01-01,AB,1323,00,
,01/01/99 1:00:00 AM,Unit under test,Fads
87-9999-01-01,AD,6789,00,
,01/02/99 3:00:00 AM,Unit failed test,Retest
```

The is a typical example for using line input mode (#) and double trailing @@s.

```
data t1;
length hd1-hd8 $25 ;
infile cards dsd firstobs=2;
input #1 hd1-hd4
      #2 hd5-hd8;
```

```
cards;
hd1,hd2,hd3,hd4,hd5,hd6,hd7,hd8
87-1234-01-01,AB,1323,00,
,01/01/99 1:00:00 AM,Unit under test,Fads
87-9999-01-01,AD,6789,00,
,01/02/99 3:00:00 AM,Unit failed test,Retest
;
```

❶ firstobs =2 will input data from line 2. dsd option is a special option for a comma delimiter file.

❷ Line pointer controls tell SAS to read hd1 – hd4 from line 1 and hd5 – hd8 from line 2.

```
data t1(drop=dummy);
length hd1-hd8 $25 ;
infile cards dsd firstobs=2;
input hd1-hd4 dummy $ hd5-hd8 @@; ❶
cards;
hd1,hd2,hd3,hd4,hd5,hd6,hd7,hd8
87-1234-01-01,AB,1323,00,
,01/01/99 1:00:00 AM,Unit under test,Fads
87-9999-01-01,AD,6789,00,
,01/02/99 3:00:00 AM,Unit failed test,Retest
;
```

❶ double trailing @@ mode. One needs an additional variable because it has a couple 's when a line wraps.

Here is a slightly different request. In this example it is easy to use a line pointer control if one is only interested in the first three fields for each record.

```
data t1;
length hd1-hd3 $25 ;
infile cards dsd firstobs=2;
input hd1-hd3
#2; ❶
cards;
hd1,hd2,hd3,hd4,hd5,hd6,hd7,hd8
87-1234-01-01,AB,1323,00,
,01/01/99 1:00:00 AM,Unit under test,Fads
87-9999-01-01,AD,6789,00,
,01/02/99 3:00:00 AM,Unit failed test,Retest
;
```

❶ It tells SAS to input the first three fields from line 1 and to skip line 2.

Case 8

One is trying to read a ASCII data file into a SAS data set. The file looks like this,

LINE	ID	VAR_NAME	TEXT
001	111112	Mein83_1	some text
002	213455	Mein83_2	other text
003	135462	Mein83_3	more text
004	351842	BESOND	complications

```
005 no money
006 critical value
007 undetermined origin
008 321682 BESOND text
009 321654 Wohn9we Buspark
```

In a standard case, the first six-character is an ID, the second collection of characters is a variable name and after some empty space there comes some text.

Because of errors, the TEXT field is wrapped at lines 005 and 006. So the text from line 005 to 007 belong to case 351842 (line 004).

The problem is that one does not know when wrapped lines occur, neither how many there will be. The nature of the case suggests that one should read a record, hold it and modify it if the next record is pure text, and then output it just before another standard case begins. Here are SAS codes.

```
data t1 ( keep = id var_name text ) ;
retain id var_name text; ❶
infile cards eof=eof truncover ;
input othtext $100. @@; ❷
if verify ( substr(othtext,1,6), '0123456789' )
eq 0 then do;
if _n_ ne 1 then output; ❸
input @1 id $char6. var_name $ text $200.;
end;
else do;
text=trim(text)||' '||trim(othtext); ❹
end;
return ;
eof:
output ; ❺
return ;
cards ;
111112 Mein83_1 some text
213455 Mein83_2 other text
135462 Mein83_3 more text
351842 BESOND complications
no money
critical value
undetermined origin
321682 BESOND text
321654 Wohn9we Buspark
;
```

❶ Variable values of ID VAR_NAME TEXT will be retained at the top of next iteration.

❷ It holds line position after it reads in the value of OTHTEXT.

❸ If first six-character is account id, it implies two things. One is the previous observation being ready to output. The other is that a new record should read into PDV.

- ④ If first six-character is not account id, it implies it is a wrapped text from the previous record.
- ⑤ It outputs the last observation in PDV.

Case 9

One has a huge dynamic text file of subject profile with a size of about 80 megabytes. On each page there is a page number on upper right corner. Below is an example:

```
( TITLE STARTS AT THIS LINE )          1
-----
( dynamic lines of text)
XXXXXXXXXX
XXXXXXXXXX
-----
(SOME LINES OF FOOTNOTES)

( TITLE STARTS AT THIS LINE )          2
-----
( dynamic lines of text)
XXXXXXXXXX
XXXXXXXXXX
-----
xx                xx
-----
( SOME LINES OF FOOTNOTES )

( XXX XXXX XXXXXX )                    999
-----
-----
(SOME LINES OF FOOTNOTES)
```

One wants to detect page 999 because it does not have body text in between the two lines.

Assume that there is at lease a blank between a page number and a title. If a page contains no body text, then there is no blank line between two dash lines. Here is the logic. If one sees the page number on upper right corner, then one checks the following two lines.

```
data t1(keep=page);
  infile cards truncover;
  input #1 x $200. ;
  y=scan(reverse(x),1);
  page=reverse(y);
  if indexc(y,'1234567890') ne 0 then do;
    input #2 line2 $50.
         #3 line3 $50.;
    if line2=line3 then output;
  end;
cards;
( TITLE STARTS AT THIS LINE )          1
-----
( dynamic lines of text)
XXXXXXXXXX
XXXXXXXXXX
```

```
-----
(SOME LINES OF FOOTNOTES)

( TITLE STARTS AT THIS LINE )          2
-----
( dynamic lines of text)
XXXXXXXXXX
XXXXXXXXXX
xx                xx
-----
( SOME LINES OF FOOTNOTES )

( XXX XXXX XXXXXX )                    999
-----
-----
(SOME LINES OF FOOTNOTES)
;
run;
```

Conclusion

It is hoped that these nine examples would give some highlights to master the functions of trailing @,@@ and line control pointer #. Understanding these functions may help you to improve your SAS skills. The more one practices, the more one would gain.

References

- [1] SAS® Language: Reference Version 6, First Edition, Cary, NC: SAS® Institute Inc., 1990, p399.
- [2] SAS® Language: Reference Version 6, First Edition, Cary, NC: SAS® Institute Inc., 1990, p398.
- [3],[4] SAS® Language: Reference Version 6, First Edition, Cary, NC: SAS® Institute Inc., 1990, p16.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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

Authors:
Jingren Shi
Syntel Inc.
jshi2@ford.com

Shiling Zhang
The Department of Mathematics
Wayne State University
shiling@math.wayne.edu