Some Not Widely Known or Sometimes Poorly Understood Features of Base SAS® Software
John R Whittington, Mediscience Services Ltd., Buckingham, UK

ABSTRACT
A few simple aspects of Base SAS are discussed which seem to be unfamiliar to even very regular users of SAS. In particular, the 'special format' of output available from PROC PRINT, the use of the colon 'informat modifier' and various aspects of SAS 'variable lists'.

INTRODUCTION
The scope and versatility of Base SAS software is such that very few of even very regular users can ever achieve a full knowledge and command of all facets of the language. Observation of discussions on internet newsgroups has indicated that there are some simple facets of the language which seem to have escaped some users. A few recently observed examples, all of which were once unknown to or misunderstood by the author, are discussed here in a whistlestop tour.

SPECIAL FORMAT OF PROC PRINT OUTPUT
Many users, like the author, have wasted time writing DATA step code to achieve a desired format of output from PROC PRINT in situations of hierarchical data when this facility actually exists as an intrinsic feature of the procedure.

Inclusion of a BY statement in PROC PRINT normally results in a separate block of output for each BY group, each with column headings and with the BY variables indicated by a title line, but not included in the data listing - for example:

```
SUBJECT=1 DAY=0
OBS   HOUR   VAR1  VAR2  VAR3
  1    0     36    10    62
  2    12    74    32    56
  3    24    83     7    46
SUBJECT=2 DAY=2
OBS   HOUR   VAR1  VAR2  VAR3
  4    0     27    57    11
  5    12    18    59    86
  6    24    72    23    9
...... etc.
```

However, if PROC PRINT includes both ID and BY statements, and if the variable specifications for those two statements are identical, then the procedure output automatically adopts this special format, as illustrated by the following example (assuming data sorted BY subject day):

```
proc print data=test;
   id subject;
   by subject;
run;
```

gives output:

```
SUBJECT  DAY  VAR1  VAR2  VAR3
   1    0     36    10    62
   2    74    32    56
   4    83     7    46
   6    27    57    11

   2    0     18    59    86
   2    72    23     9
   4     7    85    49
   6    73    55    19

   3    0     70    51     2
   2    76    20    14
   4    48    53    72
   6    18    56    70
```

Similarly, for a higher degree of hierarchical data structure (assuming data sorted BY subject day hour):

```
proc print data=test;
   id subject day;
   by subject day;
run;
```

gives output:

```
SUBJECT  DAY  HOUR  VAR1  VAR2  VAR3
   1    0     0     36    10    62
   2    74    32    56
   4    83     7    46
   6    27    57    11

   1    2     0     27    57    11
   2    18    59    86
   4    72    23     9

   2    0     0     7    85    49
   2    73    55    19
   4    70    51    2

   2    2     0     76    20    14
   2    48    53    72
   4    18    56    70
```

The same technique can be used for any number of ID/BY variables, provided always that the variable specifications are identical for both.
THE COLON INFORMAT MODIFIER

The function of the ‘colon informat modifier appears to be a very common source of confusion, in particular because it effectively switches the reading of a line of input data temporarily out of ‘list input mode’. In list input with no explicit informat, all characters read from the source file are considered to be part of the current variable until a space or other specified delimiter is encountered, with, where applicable, truncation of any characters beyond the length of that variable (specified in a LENGTH statement, or the default); reading of the next variable then commences after the delimiter, even if the data does not fill the variable or has been truncated to fit the variable length.

In the presence of an informat, however, reading of data no longer behaves in this ‘list input’ mode whilst reading the variable to which the informat is associated. Consider the following example of a DATA step which attempts to use list input to read character variables with informats specified in the INPUT statement:

```plaintext
data _null_;  
infile cards;  
input A $8. B $5.;  
put A= B=;  
cards;  
POTATOES CHIPS  
;  
run;
```

This results in the correct assignment A=POTATOES, but B is assigned the value CHIP; reading of the second variable starts in the character position immediately after the 8 characters specified by the $8. informat and reads exactly 5 characters as specified by the $5. informat - hence what is read is <space>CHIP, the final character not being read. If the data line is changed to:

```
FRIES CHIPS
```

then the assignments are A=FRIES CH and B=IPS. Even if one specifies an explicit delimiter, the same still happens, the delimiter being treated simply as a data character. Hence:

```plaintext
data _null_;  
infile cards dim=.;  
input A $8. B $5.;  
put A= B=;  
cards;  
FRIES,CHIPS  
;  
run;
```

results in A=FRIES CH and B=IPS.

The presence of an informat in the INPUT statement therefore effectively terminates ‘list input thinking’ for the number of characters specified by the informat. Reading of data for the next variable will then recommence with the very next character. If that next variable also has an associated informat, then the character following the last one read for the previous variable will be taken as the first character of the next variable. If there is no informat associated with the next variable, then behaviour reverts to ‘list input’ behaviour and commences with the first non-blank character ....

data _null_;  
infile cards dim=.;  
input A $8. B $;  
put A= B=;  
cards;  
POTATOES CHIPS  
;  
run;
```

will read the data, as intended: A=POTATOES B=CHIPS - but will, of course, again go wrong if the data value read for A does not have exactly the number of characters specified by the informat associated with variable A.

Precisely the same problem occurs with date informats if the data is written in a form which does not always have precisely the correct number of characters for a specified informat.

```plaintext
data _null_;  
infile cards dim=.;  
input A DDMMYY8. B $;  
put A= DATE7. B=;  
cards;  
09/03/96 CHIPS  
;  
run;
```

correctly gives A the date 09MAR96 and B=CHIPS. However, if the date is present in the data as 9/3/96, although the date is read correctly into variable A, the DDMMY8. informat results in two extra characters being read from the source data, so that reading for variable B commences with the ‘H’; the result is B=HIPS.

The simple solution to all these problems is the colon ‘informat modifier’. By inserting a colon between variable name and its informat in the INPUT statement, behaviour reverts to ‘list input’ reading immediately following the final character of the data read for that variable, even if the full number of characters required by the associated informat have not been read. Hence:

```plaintext
data _null_;  
infile cards;  
input A :$8. B :$5.;  
put A= B=;  
cards;  
POTATOES CHIPS  
;  
run;
```

will correctly result in A=POTATOES, and B=CHIPS. Similarly, if the data line is changed to:

```
FRIES CHIPS
```

then, again correctly, A=FRIES and B=CHIPS, even though the number of characters in the data for A were not enough for the $8. informat. Finally, in terms of date variables:

```plaintext
data _null_;  
infile cards dim=.;  
input A DDMMYY8. B $;  
put A= DATE7. B=;  
cards;  
9/3/96 CHIPS  
;  
run;
```

will assign the desired date to A and give B=CHIPS, as desired, even though the form of date in the input data does not have enough characters to fully satisfy the DDMYY8. informat.
There is an alternative approach to the use of the colon modifier. If an informat is specified in an INFORMAT statement, rather than in the INPUT statement, then data input behaves just as if there were a colon modifier associated with that informat in an input statement. Hence, for example, the following code gives the required behaviour, despite the 'short' style of date in the data:

```plaintext
data _null_;  
informat A DDMYY8. B $;  
infile cards dim=10;  
input A B:  
put A= DATE7. B=;  
cards;  
9/3/96 CHIPS  
;  
run;
```

It is, in fact, very unusual to have situations in which the presence of the colon modifier will do any harm, yet without the colon modifier there are countless opportunities for unintended interpretation of the input data. This is particularly true when one realises that 'list' (as opposed to 'formatted') input is most commonly used in situations of unpredictable data length - the very situation in which the absence of colon modifiers can cause havoc. It therefore can be argued that programmers should make routine use of colon modifiers with explicit-length character and date in formats, omitting the colons only if they ever come across a situation in which their use could result in harm.

**SAS VARIABLE LISTS AND VARIABLE ORDER IN THE PROGRAM DATA VECTOR**

Most SAS functions which require variable names as an argument will accept a 'variable list', which can take one of several forms. The most basic form consists of an explicit list of variable names, separated only by spaces (never commas, which indicate separate arguments), for example:

apple orange pear grape banana

When there are many variables to be listed, this method can be tedious, in which case some more convenient options exist. In the case of variables with a common stem followed by a numeric suffix (e.g. var1 var2 var3 .... var99), one can specify a group of these variables by use of a 'numbered range' list, simply specifying the first and last member of the group required, separated by a single dash/hyphen, for example:

var1-var99
var7-var23

More than one such list can be included within a variable name list, such as:

var1-var6 var9-var13 var32-var45

All variables which exist within the stated range will be utilised. If any do not exist (e.g. if there were no var12 in any of the above examples) a warning message is generated but program execution proceeds using all other available variables.

In the absence of numerical suffixes to variable names, groups of variables may be specified by means of a 'double dash' (a 'named range' list), although this technique requires considerably more care. For example:

orange -- grape

(NOTE: that is a double dash !)

will result in inclusion of all variables which exist in the Program Data Vector (PDV) between 'orange' and 'grape', inclusive. The need for caution arises because of the need to know the order in which the variables occur in the PDV, something which can be ascertained, for example, from PROC CONTENTS, a PUT _ALL_ statement or a PROC PRINT which has no ID or VAR statement.

The nature of programming and data sometimes means that the order of variables in the PDV is known with certainty, but this is often not the case. Furthermore, even when the order is known, it may not suit ones desire to specify a LENGTH statement of these variables using a named range list. For example, if one has the following variables, in this order in the PDV:

apple orange pear grape banana

but wants to be able to specify apple, pear and grape with a named range list, then this is not possible.

The simplest way to enforce a required ordering of variables in the PDV is to include a RETAIN statement at the start of the DATA step. For example, if one commenced a DATA step with:

```plaintext
data one;  
retain apple pear grape orange banana;
```

ten this would force that ordering on the variables in the PDV, such that a subsequent variable list (apple-grape) would specify apple, pear and grape, as required. Use of a RETAIN statement in this fashion is nearly always harmless, one of the few situations when care is needed being when there are multiple SET statements within a DATA step. A LENGTH statement at the start of a DATA step will have the same effect of forcing variables to appear in the PDV in the order specified in that statement, but is generally less convenient because a value for length does have to be specified.

As a final point, it is worth remembering that many SAS functions can either take multiple arguments or a list, but sometimes with slightly different syntax. For example:

```plaintext
total = sum(var1, var2, var3, var4);  
total = sum(of var1 var2 var3 var4);  
total = sum(of var1-var4);  
total = sum(of array());
```

and also that there are intrinsic 'special' named range lists:

_ALL_, _NUMERIC_ and _CHARACTER_

**CONCLUSION**

Many very simple features of SAS escape even very regular users. A few often overlooked points have been discussed briefly.

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

John R Whittington  
Mediscience Services Ltd.,  
Twyford Manor,  
Twyford, Buckingham MK18 4EL, UK  
Voice: +44 1296 730225  
Fax: +44 1296 738893  
e-mail: johnw@mag-net.co.uk  
100517.3677@compuserve.com