The Way of the Semicolon
-or-
Three Things I Wish I Had Known Before I Ever Coded One

Patricia Hettinger, Oakbrook Terrace, IL

ABSTRACT

Learning SAS or teaching it to someone else can be very difficult. The author has found that understanding three main aspects of SAS® very helpful. These are of course, the semicolon, the physical nature of a SAS data set and the two major data types. The intended audience is for those who are new to SAS or new to training others in it.

INTRODUCTION

This paper will address three key areas of difficulty the author has seen again and again. These are punctuation confusion, SAS data set properties and the difference between the two basic variable types, alphanumeric and numeric. Getting a handle on these three areas will make developing your analytical skills and technique much easier.

We will be talking about data sets and procedures. Data sets contain the data you want to analyze and SAS procedures are programs provided by SAS to do this analysis. There are only two examples of procedures in this paper, PROC CONTENTS and PROC PRINT, one to illuminate data set structure and the other to demonstrate the use of the semicolon.

THE SEMICOLON AND OTHER PUNCTUATION

SAS punctuation is different from standard English. It's also quite different from the most common programming/scripting languages used today like .NET, Java and C. The 'Way of the Semicolon' was named this for a reason. The semicolon dominates SAS just as the period dominates English writing. Let's take a look at how SAS might interpret this famous soliloquy:

To be, or not to be--that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune
Or to take arms against a sea of troubles
And by opposing end them. To die, to sleep-

The periods are replaced by semicolons. Uncoupled quotes like 'tis aren't allowed so either we couple it as below or take it out altogether. The comma and dashes are out too:

To be or not to be that is the question**;**
Whether **'**tis' nobler in the mind to suffer
The slings and arrows of outrageous fortune **;**
Or to take arms against a sea of troubles
And by opposing end them**;** To die to sleep**;**

Probably half of the errors you find in the log are due to the semicolon being in the wrong place. This is usually for one of two reasons – either the statement has ended and you forgot the semicolon or the statement had options that shouldn't have had the semicolon first but did. Statements can be procedures, data steps (more on this in the next section), functions, etc. Procedures can have several options that can make it confusing as to where to put the semicolon. Let's take the PROC PRINT procedure which prints out data set variables for example

```
Proc print data=something noobs; (noobs is an option to not print observation numbers)
Proc print data-something noobs      - no semicolon, error
Proc print data=something; noobs    - semicolon too soon, error also.
```

This rhyme might help:

Semicolon at statement end;
SAS is your friend;
Before the options though;
The step won't flow;
When in doubt put it in;
If error take it out again;

STEPPING OUT WITH THE DATA SET

Just as the action of the play is not possible without the characters, SAS analysis is not possible without SAS data. And SAS data is not possible without variables and variable definitions. SAS analytical and display actions are referred to as PROCEDURES (the PROC PRINT punctuation example is one) and the data as a DATA SET. Excel spreadsheets have rows and columns. SAS data sets have observations and variables..

The essential pieces of a data step are the DATA statement and name of data set to be created, ended with a semicolon (of course). Declaration of characteristics and a method to get data in are also required. We will discuss three beginner techniques, two actually setting the values in the step and one taking the values from a data set already created.

So how might we define our moody prince? We know this from Shakespeare's immortal play:

> His name is Hamlet
> His title is Prince of Denmark
> His mother is Gertrude
> His father was Hamlet
> A relative is Claudius, the current King
> He's a friend of Horatio
> His sanity is debatable and has been for centuries

We don't yet have Hamlet in a data set so we'll have to put him there. One way is the datalines method where we put the data in by position. Here we're expecting everything to start in position 1 (Name). That incoming field is 6 characters long ($6.) so we're expecting the title variable to follow after the Name and a separating space. Here is the first place you will see a period in SAS, at the end of a format Character formats begin with $ and numeric do not. There are no numeric fields being read in this particular step:

```
data hamlet;
attrib             /*attrib statement lets you specify different lengths
                           and even formats from the data coming in*/
Name                       format=$20.      /*character length of 20*/
Title                      format=$20.
Mother_name                format=$20.
Father_name                format=$20.
relative                   format=$20.
Friend                     format=$20.
Sanity_evaluation          format=$10.
;
Input                      /*Input statement describes data coming in*/
@1 Name                            $6.      /*incoming is character length of 6*/
title                              $18.
Mother_name                        $9.
Father_name                        $7.
relative                           $9.
friend                             $8.
sanity_evaluation                  $10.
;   /*datalines statement following is dependent upon actual position in data line*/
datalines;
Hamlet Prince of Denmark Gertrude Hamlet Claudius Horatio Debatable
;
```

The author prefers creating the variables by name so that the actual position is irrelevant:

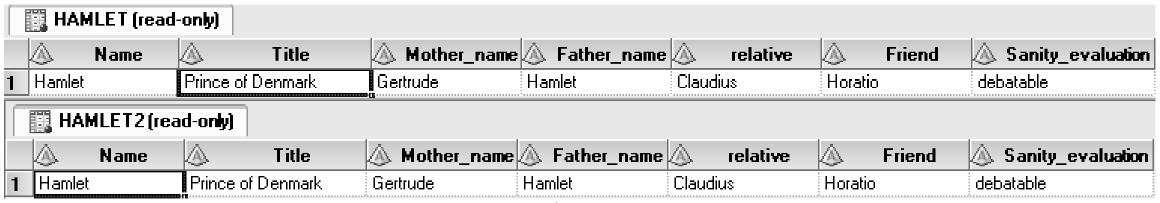
```
data hamlet;
attrib              /*attrib statement lets you specify different lengths
                            and even formats from the data coming in*/
Name                                format=$20.
Title                       format=$20.
Mother_name                 format=$20.
Father_name                 format=$20.
relative                    format=$20.
Friend                      format=$20.
Sanity_evaluation           format=$10.
; /*we don't need the position here*/
Name='Hamlet'; title = 'Prince of Denmark'; mother_name='Gertrude';
father_name='Hamlet'; relative='Claudius'; friend='Horatio';
sanity_evaluation='debatable'; output;
```

If we wanted two hamlets, we could use the SET statement to create another one::

```
DATA hamlet2;
SET hamlet;
```

Both DATA and SET are statements so they must end with a semicolon.

The end result is two identical data sets, HAMLET and HAMLET2:



HAMLET (read-only)

	Name	Title	Mother_name	Father_name	relative	Friend	Sanity_evaluation
1	Hamlet	Prince of Denmark	Gertrude	Hamlet	Claudius	Horatio	debatable

HAMLET2 (read-only)

	Name	Title	Mother_name	Father_name	relative	Friend	Sanity_evaluation
1	Hamlet	Prince of Denmark	Gertrude	Hamlet	Claudius	Horatio	debatable

Figure 1

Figure 1 shows the view from SAS Enterprise Guide, SAS's GUI interface. It looks something like an Excel spreadsheet here but it's really SAS's proprietary internal format. If we looked at a SAS data set created on Windows XP with a text editor, we might see

```
DATASTEPName$   Title   $   Mother_name $   Father_name $   relative$   Friend  $   Sanity_evaluation   $
üÿÿˆ   □   □     - more unprintable characters.
```

Woe to you if you try updating anything in a text editor. You're likely to corrupt your data set and make everything in it unusable. Bad for any data set you're like to use more than once. Use SAS to read and update SAS data sets.

Fortunately there is a procedure named PROC CONTENTS that gives us this information in text-readable form. Key the statement: PROC CONTENTS data=hamlet**;** This syntax is used for most SAS procedures: PROC whatever DATA=whatever;

Output:

```
The CONTENTS Procedure

A) Data Set Name          WORK.HAMLET                      Observations          1
B) Member Type            DATA                            Variables             7
C) Engine                 V9                               Indexes               0
D) Created                Thu, Sep 08, 2011 09:38:37 PM   Observation Length    130
E) Last Modified          Thu, Sep 08, 2011 09:38:37 PM   Deleted Observations   0
F) Data Representation    WINDOWS_32
G) Encoding               wlatin1  Western (Windows)


                    Engine/Host Dependent Information

H) File Name              C:\DOCUME~1\PATHET~1\LOCALS~1\Temp\SAS Temporary
                 Files\_TD256\Prc2\hamlet.sas7bdat
I) Release Created        9.0101M3
J) Host Created           XP_PRO
```

A) Name of the data set just created and number of observations. Just 1 because only Hamlet is in the data set. The location of HAMLET is WORK, which means it will disappear once your session is over
B) Type of member DATA with 7 variables (see part two for list)
C) Engine – which version of SAS created this?
D) Date created and physical length of observation (20*6 =10) = 130
E) Date Last Changed – here the same as created date
F) How is this data coded? UNIX and OS/390 files would give different data representations
G) Encoding is standard windows
H) File name – this is a work file so the path is temporary
I) Release created – 9.01
J) Where was data set created? Here XP_PRO

```
Alphabetic List of Variables and Attributes

#   Variable              Type   Len   Format

3   Father_name           Char    20   $20.
5   Friend1               Char    20   $20.
6   Friend2               Char    20   $20.
2   Mother_name           Char    20   $20.
4   Other_relative        Char    20   $20.
7   Sanity_evaluation     Char    10   $10.
1   Title                 Char    20   $20.
```

List of variables gives name, type (either Char or Num), length and the format.

If you want to keep Hamlet around past your session, you must specify a permanent location for him. The file name in the PROC CONTENTS output above (H) makes the temporary location very clear. LIBNAME lets you give a short-cut for the location so you don't need to remember exactly where it is all the time. In the real world, we might have Hamlet at Drury Lane in Oakbrook Terrace, IL. We could use the LIBNAME statement to give a short-cut for this address:

LIBNAME druryoak 'Drury Lane Oakbrook 100 Drury Lane Oakbrook Terrace, Illinois 60181'**;**

To refer to the Hamlet at Drury Lane, we use the period: DATA druryoak.hamlet; This marks the second time we have used the period. The first was to specify the variable formats which were all character: $20. or $10. A numeric variable would be 20. or 10., no dollar sign. However, the underlying format for a numeric value is length of 8, no matter which format you specify.

SAS automatically keeps track of several things about data sets. In our hamlet2 creation code above, we only SET our original hamlet data set. If we had combined two or more, we could take advantage of this to identify which data set it came from, whether this was the first or last hamlet entry and even the number of observations in each one. This code is a bit more complex but there are so many uses for this:

```
data hamlet2;
set hamlet (in=a)  /*identify this as 'a'*/
hamlet(in=b)               /*identify this as 'b'*/
hamlet(in=c);              /*identify this as 'c'*/
BY NAME;                   /*should work because the source is repeated
                           and all the Name values are 'hamlet'*/
/*if this is the FIRST occurrence of 'hamlet' set the entry_order value*/
if first.name then entry_order='FIRST  ';
/*if this is the LAST occurrence of 'hamlet' set the entry_order value*/
else if last.name then entry_order='LAST';
/*anything else will be considered in the middle*/
else entry_order='MIDDLE';
if a then dataset_order='A'; /*SAS knows which data set this observation
came from*/
else if b then dataset_order='B';
else dataset_order='C';
run;
```

The values for entry_order and data set order end up being:

```
Obs     Name        entry_order     dataset_order
1       Hamlet      FIRST           A
2       Hamlet      MIDDLE          B
3       Hamlet      LAST            C
```

ABOUT SAS VARIABLES

SAS can read and write data in a large number of formats, even the mainframe packed decimal. However, within a SAS data set, there are only two types, character and numeric. The variable name cannot be more than 32 characters long. It also cannot start with a number or have special characters besides an underscore unless you surround the name with single quotes and an ‘N’ like ‘2MORTGAGE’N. Although you can create variables named like this, it’s not recommended because you will always have to refer to them with the quotes and ending N.

To read data into a SAS data set, you need an INFORMAT. To write it out, you need a FORMAT. Most of the informats have a corresponding format. Two exceptions are the informat anydtdte*n*. and the format Z*n* where *n* is the length. The former does a good job of reading in almost any date with a four-digit year but if you want to write a date out, you need a specific format. The Z*n*. Format writes out numeric data with leading zeros. There is no corresponding informat because SAS (and really any program that does math) doesn’t care about them. You might care if you have account numbers with leading zeros (more common than you might think) or social security numbers which may also have significant leading zeros. Whether either should be numeric format is debatable but since so often they are, you need to specify leading zeros when writing them out.

To demonstrate, here’s a data step that setting two numeric variables with different formats to the same value. Notice the IF statement that sets another variable equality_status:

```
DATA numformats;
attrib account_num format=16.
account_num2 format=Z16.;
account_num =0123456789012345;
account_num2=0123456789012345;
IF account_num = account_num2 THEN equality_status = 'EQUAL';
```

The appearance of these would be different when output but the values are still the same:

```
account_num             account_num2            equality_status
123456789012345         0123456789012345        EQUAL
```

If you changed the format of account_num and account_num2 to character, but did not put the values in surrounding quotes as in the following code, neither your account_num nor account_num2 would have the leading zeros but they would still be equal to each other:

```
data numformats;
attrib account_num format=$16.
account_num2 format=$16.;
account_num =123456789012345;
account_num 2=0123456789012345;
IF account_num = account_num 2 THEN equality_status = 'EQUAL';
```

```
account_num          account_num2         equality_status
123456789012345      123456789012345      EQUAL
```

But if you put the values in quotes, they will not evaluate to equal:

```
data numformats;
attrib account_num format=$16.
account_num2 format=$16.;
account_num ='123456789012345';
account_num 2='0123456789012345';
IF account_num = account_num 2 THEN equality_status = 'EQUAL';
```

```
account_num          account_num2         equality_status
123456789012345      0123456789012345
```

In fact, equality_status will be blank. That's because we did not code for the condition 'not equal'. Therefore, the equality_status is NULL or unknown. This looks like an empty space for character data in SAS. The default display for an unknown or missing numeric value is '.', the third time you will see a period in SAS. As with most programming languages and data formats, the end result of a calculation with a missing value is a missing value itself.

2+2=4 but 2+2+**.**=**.** **2+2+Missing=Missing**

SAS DATES

SAS dates are numbers too. The range is from -138,061 to 2,936,547. Figure 2 demonstrates this range. Unlike our western reckoning of B.C. and A.D., there is a day 0.

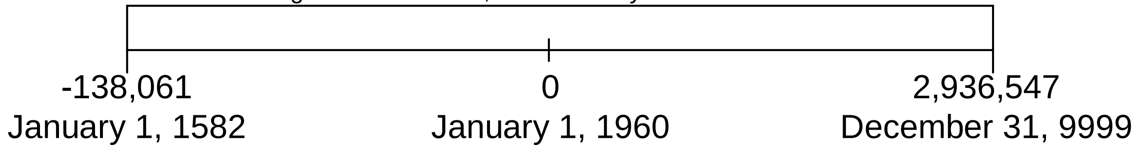


Figure 2

You can put numbers outside this range into a date format but you will see ***** for overflow. As you can see, SAS never had a Y2K problem but it might have a Y10K problem if any of us are around that long.

You refer to a SAS date by either it's number or a literal in the format 'ddmmyy'd. November 7, 2011 would be either 18938 or '07NOV2011'd. Although still somewhat clumsy, the later is certainly easier for a human being to remember.

You can do calculations with SAS dates too. To find the number of days between two dates, simply subtract them: To advance your date by some number of days, simply add that number to the date. There are two functions that manipulate dates that everyone should know. These are INTNX and INTCK. INTNX advances or retreats a date by a given interval and INTCK gives the number of intervals between two dates. Unfortunately, these function names don't appear to be acronyms so remembering their names might be tricky. The general syntax for INTNX is **INTNX**(*interval*<*multiple*><*.shift-index*>, *start-from*, *increment*<,*alignment*>) and INTCK is **INTCK**(*interval*, *from*, *to*) . The tables below in figure 3 and 4 should give a good idea of how these are used:

A	**B**	**C**	**D**	**E**
Getting system Date	How many days ago was July 4, 2011?	beginning day of previous month	End day of previous month	Exactly One month ago
thedate=today()	daysago = thedate-'04jul2011'd	lastmonthbeg = INTX('month',thedate, -1, ' beginning');	lastmonthend = INTX('month',thedate, -1, 'end');	lastmonthexact = INTX('month',thedate, -1, 'same');
thedate	daysago	lastmonthbeg	lastmonthend	lastmonthexact
7/19/2011	15	6/1/2011	6/30/2011	6/19/2011

Figure 3

A – we can assign the system date by using the today() function. Here we are setting a variable called thedate.
B – to find a date days in the past or in the future, simply add or subtract one date from another.
C – Many times we are running reports or analysis for the previous month. We can use the INTNX function to find the first day of the previous month by putting 'beginning' in as the fourth parameter. The -1 tells us to go to the previous month.
D – To find the end day of the previous month, use 'end' as the fourth parameter.
E – To find what we would consider exactly one month ago, use 'same' as the fourth parameter

F	**G**	**H**
How shopping days until 12/25/2011	How many months away is August 1, 2011	How many months away is July 31,2011
shoppingdays = '25dec2011'd - thedate;	monthinterval1 = intck('month',thedate,'01aug2011'd);	monthinterval2 = intck('month',thedate,'31jul2011'd);
shoppingdays	monthinterval1	monthinterval1
159	1	0

Figure 4

F - Subtract one date from another to find the number of days between them
G – The number of months is the next whole number after the interval has been crossed (8/1/2011 crosses an interval as far as July 19, 2011 is concerned. Thus the number of months between the two is 1
H. July 31, 2011 does not cross an interval coming from July 19, 2011 so the number of months between the two is 0

CONCLUSION

Remembering these three basic points about punctuation, SAS data set characteristics and the two types of SAS variables should give a good foundation for learning SAS before you even turn on the computer. It should also aid in teaching others.

ACKNOWLEDGEMENTS

Thanks to Joe and Paul Butkovich for your encouragement and support.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. The author is often on the road but can be contacted at

Patricia Hettinger
Email: patricia_hettinger@att.net
Phone: 331-462-2142