Why Does SAS® Run Clockwise?
Rob Krajcik, Bristol-Myers Squibb, Wallingford, CT

Abstract

With apologies to author David Feldman, every programming language has its quirks -- things that are not considered bugs or defects but just behave in an unexpected way. And SAS is no Exception. Starting with a brief history of the SAS compiler and Macro processor, The author will take you through some of the quirks (or traps) and undocumented features of SAS 9.1 that you may also encounter.

Introduction

Originally, "SAS" stood for "Statistical Analysis System", however, the applicability of SAS has grown far beyond that. Besides statistics, elements of the SAS system also include web enablement, data mining, data warehousing, business intelligence solutions, as well as a comprehensive system for data management, analysis, and reporting. Today "SAS" is used to refer to both the software product and the company behind it. By the way, "The SAS" refers to a regiment of the United Kingdom's Special Forces.

Q: Why do clocks run Clockwise?

A: Mechanical clocks with hands were originally built to imitate the path of a sundial shadow. In the northern hemisphere, where sundials were first used extensively, the shadow on a sundial would move around its face in a "clockwise" (left to right) direction. In the southern hemisphere, the shadow on a sundial would move right to left.

About Quirks

What is a quirk? To a logician the following statements may seem quirky:

\[ A = \text{not}(A) \]
\[ X = X + 1 \]

When you understand the "not" is a function, and the equals sign to the left of X+1 is really an assignment statement, they don't seem so quirky.

By "Why Does SAS Run Clockwise" I mean, Why does SAS behave the way it does?

Every language has its quirks, and SAS is no exception. These are not bugs or software defects, just things that seem to act in a non-intuitive way. You may hear speakers talk of SAS "traps" and "gotchas".

These are the same as what I am talking about here. Some of these come about from programmers who have an intuitive feel for how SAS works, but perhaps hadn't read the manual in detail. Perhaps it was undocumented.

Quirky things can happen when using the MERGE statement. There have been many presentations on merge traps by notables such as Bob Virgile and Ian Whitlock. And Malachy Foley seems to have spent half his career researching them! So I won't present any merge traps here. But here is a tip: Keep it short
and simple. And do your If/Then/Else and Where processing somewhere else. You are far less likely to get in trouble this way:

```sas
data <merged>;
  merge <aa_data in=ina> <bb_data in=inb>;
  by <by_vars>;
  if ina /* and/or inb */;
run;
```

A Brief History

It might also help to understand how SAS (both the software and the company) evolved over time. So I digress for a moment...

In the Sixties, the computer business had just burst on the world in a big way, after a new Texas invention, the transistor, had finally made computers a practical proposition. Before the transistor it took a building full of vacuum tubes to do any complicated calculations electronically. The arrival of the transistor changed all that. A powerful computer small enough to fit into a large room could now be built... and an industry was born.

IBM, with its tabulating machines, had already been in the computer business for thirty years. They were already so well entrenched in the calculator business that it would be next to impossible to unseat them. Several tried. Besides IBM, there were lots of players in the build-a-new-computer game for a short while... NCR, Control Data, General Electric, RCA, Univac, Burroughs, and Honeywell. After RCA and GE dropped out of the market in the early 1970's, IBM's remaining competitors were often referred to as the "BUNCH" Burroughs, Univac, NCR, CDC, and Honeywell.

IBM had the largest share of the mainframe market at the time, and was licensing it's mainframes for next to nothing to universities. North Carolina State University, where Anthony J Barr and James Goodnight were students in the late 1960s, had at least one IBM system. So it made perfect sense for the two of them to develop SAS on an IBM machine and to later form a partnership with IBM.

When SAS first entered the market in 1976 it had many competitors. One of SAS's strengths was analyzing experiments with missing data, but the data step language was the one feature that made SAS stand out head and shoulders over its competition!

Anthony J Barr was the architect of the 1976 version of the Statistical Analysis System. The 1976 version was a comprehensive system for statistics, data management and report writing. According to his website, [www.barrsystems.com](http://www.barrsystems.com), he designed and implemented all of the programming language, data management, report writing, and systems areas.

PL/I (Programming Language 1) was a large, complex block-structured language invented by IBM, and first released in 1964 in conjunction with the influential System/360 line of computers. PL/I was an attempt to combine the best features of Algol (program structure, semantics), FORTRAN (calculations), and COBOL (data structuring, I/O) into one new, all-purpose language. The result was a very complex language, but one that did serve most programming purposes quite well.

It is no surprise then, that the SAS Data Step language closely resembled PL/I. Unlike PL/I however,, the Data Step section of a SAS program, assumes a default file structure, and automates the process of identifying files to the operating system, opening the input file, reading the next record, opening the output file, writing the next record, and closing the files.
This allowed the user/programmer to concentrate on the details of working with the data within each record, simplifying the overall programming effort. SAS also has added features that PL/I does not, such as: formats, automatic variables, and an implied data loop with an implied output statement.

At that time, punch cards for input were in use. They were 80-columns wide and often used columns 73-80 for sequence numbers. Printers were the impact variety having a maximum width of 133-characters, the first one used for carriage control and pagination. To print in bold, you needed to tell the printer to overprint (hold the current line and print over it -- Options OVP). Options S=72 was often used in case the user had sequence numbers on their cards. And there were no lower case characters on those card punches! Block-Style comments (* *) never started in column 1 as IBM's JES2 job entry system would confuse it with an end-of-file marker. INFILE / CARDS is still used today (DATALINES came later). FORTRAN and PL/I programmers would use the letters I,J,K... for loop counters as any variable name beginning with the letters I-N defaulted to a 4-byte integer. “Proc Printto Unit=21” referred to FORTRAN filename FT21F001. The logical operators LT, LE, EG, GE, GT, NE also came from FORTRAN.

Many enhancements were made to the SAS Data Step language over the years. In an e-mail message to the author on February 7, 2009, Rick Langston revealed that Doug Cockrell was responsible for implementing the first version of the macro processor. Up until that time there was only “MACRO xyz...%” with everything between the name and the “%” directly substituted. The old style statement macros were quickly replaced, for good reason!

Rick also informed the author that SAS's acquisition of Lattice Inc. in 1987 allowed them to gain the C expertise and compiler experience to successfully create a C implementation for PC-DOS and to move forward on the first truly portable version across all platforms. 6.06 was the first version that used common code for IBM mainframe (MVS and CMS), minicomputers (VAX, DG, and Prime), PC-DOS, and a number of Unix boxes (HP-UX, SunOS, and AIX).

Version 6.06 was also where SAS introduced things like the "In" Operator, "Where" Processing, Stored Compiled Data Step Code, Proc SQL, Indexing of SAS Data Files, and the "File Name" Statement. Many SCL (and data step) functions became available to the macro processor via the %SYSFUNC function in versions 6.09E (MVS) and 6.11 (Unix, Windows).

The Output Delivery System (ODS) and long variable names were first introduced in Version 7, a transitional release. Specifying “Options validvarname=V7” meant you intended to use more than 8 characters for your variable names. And the variable names no longer came out in uppercase by default but in “WYTIWYG” (What You Typed Is What You Got) format. The maximum length for character variables changed from 200 to 32,767 bytes. However, SAS wisely decided to leave the default length for results of character functions (such as REPEAT) to 200 bytes.

SAS 9.x introduced a whole host of new character functions, date formats, many ODS enhancements, longer names for formats and informats, a hash object that allows functionality similar to the MERGE statement without sorting data or building formats, and PERL regular expressions. But as the functionality grows and expands, so does the documentation.

That’s How it Quirks

Ever wonder why the TRANSLATE function arguments are (source, to, from) while the newer TRANWRD function arguments are (source, from, to)? Because that’s how TRANSLATE was implemented in PL/I.

I had always wondered why SAS chose BEST12 as the default numeric to character conversion format. The answer I got from SAS Technical Support was "The developer said it's his speculation that even though one could expect up to 15 digits, if you wanted a “put _all_” output (which uses Best. format) that you'd be
able to see more data in fewer lines. In those days there were punch cards and printed output on large, noisy impact printers, and using `[BEST12.]` would conserve some paper."

Here are some more examples of quirks:

Q: What is the "best" way to display all 15 digits of an irrational number?

```sas
data _null_;  
e=exp(1); pi=arcsin(-1); s2=sqrt(2);  
put e= best15. +2 pi= best15. +1 s2= best15.;  
put (e pi s2) (= best16.);  
put (e pi s2) (= best17.);  
put (e pi s2) (= best18.);  
run;
```

```
e=2.718281828459   pi=3.1415926535898  s2=1.4142135623731
A: <-BEST17.
```

Were you thinking best16. would work? Fifteen digits plus one for the dot? I was.

Q: What’s the length of a null string?

```sas
data _null_;  
string='';  
dat_len=length(string);  
mac_len=input(resolve('%eval(%length(%str()))'),2.);  
put (dat_len mac_len) (=);  
run;
```

```
dat_len=1 mac_len=0
```

A: It depends who you ask!
The data step compiler says it’s 1. The macro processor says it’s 0.
NOTE: in SAS 9.x, LENGTHN() returns 0 for a null string.

Q: Why are there only 4 observations below?

```sas
data _null_;  
set class(where=(age ge 15) firstobs=2 obs=9);  
put (name age height row) (=);  
run;
```

```
Name=Mary Age=15 Height=66.5 row=14
Name=Philip Age=16 Height=72 row=15
Name=Ronald Age=15 Height=67 row=17
Name=William Age=15 Height=66.5 row=19
```

```
NOTE: There were 4 observations read from the data set SASHELP.CLASS.
WHERE age=>15;
```

A: WHERE Processing takes precedence over firstobs and (last)obs. Here it found five observations matching the where clause. It started at the second observation and would have gone to the ninth (for a total of 8) if there were that many matching the WHERE clause.
Q: What happens when you use data-step line comments inside a macro?

```
%macro _testit(x);
  %* This is a comment;
  * %let x = 2;
  %put x = &x;
%mend _testit;

_%testit(3);
```

A: And the answer is:

```
x = 2
```

The Macro Processor executed the %let statement you thought you had commented out!

Q: How is RESOLVE different from SYMGET?

```
%let strng=%str(a string with three trailing blanks   );
data _null_;   
  length str_a str_b $50;
  str_a=resolve('&strng')||'**';
  put str_a=
  str_b=symget('strng')||'**';
  put str_b=
run;
str_a=a string with three trailing blanks*
str_b=a string with three trailing blanks   *
```

A: Besides being able to resolve macro functions as well as macro variables, RESOLVE removes trailing blanks, while SYMGET does not. RESOLVE behaves like the TRIM function with a non-macro/macro variable argument.

Other functions that will remove trailing blanks: CATS, CATT, PUTC, REVERSE, SCAN, STRIP, SUBSTR, TRIM (of course), COMPRESS (default is to remove blanks, including trailing ones), and COMPBL (which will remove all but the first trailing blank.)

Q: If datetime16. gives us a 16-character datetime value with a 2-digit year, what length do we need to get a datetime value with a 4-digit year?

```
data _null_;   
call execute('data _null_;');
call execute(' length text $21;');
do sz=07 to 20;
call execute('sz='||put(sz,z2.)||';');
call execute('text="*"||put(datetime(),datetime'||put(sz,z2.)||'.);');
call execute('put sz= z2. text=;');
end;
call execute('run;');
run;
```
Of course, you can always do it yourself with a picture format.

```sas
proc format lib=work;
    picture date_hms (default=18)
        low-high='%0d%b%Y:%0H:%0M:%0S'(datatype = datetime);
quit;

data _null_; 
    dt=datetime();
    dattm1=left(put(dt,datetime19.));
    dattm2=put(dt,date_hms18.);
    put (dattm1 dattm2) (/=);
run;
dattm1=23APR2009:14:16:50
dattm2=23APR2009:14:16:50
```

Q: Where did that format go?

```sas
data class;
    set sashelp.class;
    format height percent.;
run;
proc sort data=class;
    by name;
    format _all_; 
run;
proc contents data=class varnum; run;
```

A: The “format _all_” statement in the above PROC SORT prevented the `percent.` format from being copied to the output data set.
Q: What happens when you transpose an empty data set?

```sas
data class;
  set sashelp.class(sortedby=name);
  where age is missing;
run;
NOTE: There were 0 observations read from the data set SASHHELP.CLASS WHERE age is null;
NOTE: The data set WORK.CLASS has 0 observations and 5 variables.

proc transpose data=class out=classxp(drop=_name_);
  by _all_
  var height weight;
run;
NOTE: There were 2 observations read from the data set WORK.CLASS.
NOTE: The data set WORK.CLASSXP has 2 observations and 7 variables.
```

A: You end up with a data set containing one row for every variable in the Proc Transpose “var” statement!

Q: What’s the difference between using a Null Where versus a Null If statement?

```sas
data class;
  set sashelp.class;
  where;
run;
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.CLASS has 19 observations and 5 variables.

data class;
  set sashelp.class;
  if;
run;
```

A: A Null Where statement works!

Q: What is the difference between Where ‘0’ and If ‘0’? (From Phil Mason)

```sas
data class;
  set sashelp.class;
  where('0');
run;
NOTE: There were 19 observations read from the data set SASHELP.CLASS WHERE '0';
NOTE: The data set WORK.CLASS has 19 observations and 5 variables.

data class;
  set sashelp.class;
  if '0';
run;
NOTE: Character values have been converted to numeric values at the places given by: (Line):(Column). 33:9
```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.CLASS has 0 observations and 5 variables.

A: If you use a character variable by itself in a WHERE clause, the WHERE clause will select observations where the value of the character variable is not blank.

Q: What can happen when you store a number into a macro variable using the default BEST12. format?

```sas
data _null_;  
  dt1=datetime();  
  call symput('dt2',dt1);  
  dt2=symget('dt2');  
  dt3=.Z;  
  put (dt1 dt2 dt3) (=);  
  put (dt1 dt2 dt3) (= hex16.);  
run;
```

dt1=1563202674.3   dt2=1563202674.3   dt3=Z
dt1=41D74B251C934396 dt2=3135363332303236 dt3=Z

A: Loss of precision can result. Converting to character using a BEST17. (15 digits plus the dot plus one for quirkiness) format in the call symput instead of letting the default BEST12. conversion take place will ensure better precision. Displaying the numeric values in HEX16. format will show the real representation of those values. Unless it's missing. Missing values won't display in Hex! Yet another quirk.

Q: Why are there two quotes inside these macro comments?

```sas
%macro var_exist(data,varname);
  %* Macro Function var_exist checks if varname exists -- RJK;
  %* Returns 0 if doesn''t exist, gt 0 (varnum order) if it does;
  %let dsid=%sysfunc(open(%str(&data)));
  %sysfunc(varnum(&dsid,&varname))
  %let rc=%sysfunc(close(&dsid));
%mend var_exist;
```

A: It's a Quirk!

**The Return of John Frum**

Every February 15th, on the island of Tanna in Vanuatu (formerly known as the New Hebrides), islanders celebrate John Frum Day. On this holiest of days, devotees have descended on the village of Lamakara from all over the island to honor a ghostly American messiah, John Frum. “John promised he'll bring planeloads and shiploads of cargo to us from America if we pray to him,” a village elder tells me as he salutes the Stars and Stripes. (Smithsonian Magazine, Feb 2006)

**Cargo cult programming** is a style of computer programming that is characterized by the ritual inclusion of code or program structures that serve no real purpose. Cargo cult programming is typically symptomatic of a programmer not quite understanding the code he or she just pasted into their program from another source. If you look carefully, some of the quirks that you encounter may be the result of cargo cult programming.

Here are some examples:
If x=9
then x=3;
else x=x;
Of course, the else statement is not needed.

The following programmer did not read the section about numeric functions such as floor, int, round, or ceil.

```sas
data _null_;
x = 5.2;
y = input(x, 2.);
put y=;
run;
```
What he wanted to do was truncate the decimal. What he got was a message and a missing result.

NOTE: Numeric values have been converted to character values...

Interestingly, if he had used an informat of 10. instead of 2. in the input function, he would have gotten the result he wanted. Why?

```sas
data alpha;
set bravo;
by echo foxtrot golf hotel;
if first.echo;
run;
```
Why the long by statement? Only the variable echo is used for first. processing.

```sas
data class;
set sashelp.class;
if sex='F' then type='Female';
if sex='M' then type='Male';
else type='??';
run;
```
How many females did we end up with here? Zero?

I love this one:

```
QTC = right(trim(left(put(_6,3.))))||"(B)";
```
How about if we just use the put statement? Like this:

```
QTC = put(_6,3.) ||"(B)" ;
```

**Conclusions**

Now you have some history of how SAS began and continues to grow. Some of the quirks you may encounter are tied to the history of SAS, some are just buried in the documentation somewhere, and some you may encounter are just from folks who are sure they know how SAS works but haven’t read that chapter yet. I hope this has given you some insight into how SAS really works (and quirks).
References


SAUSAG -- The SAS Canberra Users Group for ACT. Available at http://sausag.sasusers.net/

Smithsonian Magazine (Feb 2006), "In John They Trust", available at http://www.smithsonianmag.com/people-places/10021366.html


Disclaimer

All code contained in this paper is provided on an "AS IS" basis, without warranty. The author makes no representation, or warranty, either express or implied, with respect to the programs, their quality, accuracy, or fitness for a specific purpose. Therefore, the author shall have no liability to you or any other person or entity with respect to any liability, loss, or damage caused or alleged to have been caused directly or indirectly by the programs provided in this paper. This includes, but is not limited to, interruption of service, loss of data, loss of profits, or consequential damages from the use of these programs.
Acknowledgments

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.

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

Many of the quirks, traps, and oddities presented here were probably recycled from other NESUG and local SAS user group papers and presentations.

I want to thank Jerry Le Breton (SAUSAG), Phil Mason, Ron Fehd, Ian Whitlock, Bob Virgile, Paul Dorfman, Michael Davis, Art Carpenter, Malachy Foley, Richard DeVenezia, Frank DiLorio, Rick Langston, Neil Howard, Ray Pass, Ron Cody, Kirk Paul Lafler, and many others who unknowingly gave me ideas for this presentation.

Contact Information

Rob Krajcik
Bristol-Myers Squibb
5 Research PKWY
Wallingford, CT 06492-7660

(203) 677-6125 (phone)
(203) 677-6197 (fax)

robert.krajcik@bms.com