Paper CT12

Know Your Environment

Will Greenway, Plus-Project Partnership, Leeds, UK

ABSTRACT

Often when we write SAS® programs there is a tendency to think that if we just use basic SAS statements then the program will be easily transportable across programming environments. Rarely do we take a step back and consider what operating system and SAS options we have taken for granted that may be impacting the program in the background, influencing how it was written.

For example, OPTIONS VALIDVARNAME is one of my favourite options, often overlooked but its impact can be quite large on how programs are written. Rarely is this option changed, instead programmers write for the environment they are currently in and the behaviour they see.

This paper will discuss several environment considerations and SAS options that, when accounted for, can make your SAS macros or solutions more platform independent.

INTRODUCTION

SAS programmers are very used to debugging their code, but sometimes we just can't work out why SAS behaves in one way, particularly when it acts differently when other users run the same code, even if on seemingly similar environments, or even the same environment.

The contents of this paper arose from a series of real-life questions or issues that I have come across over my programming career. Having programmed on various SAS versions and helped in the validation of new computing environments, which usually involve porting and testing code across Operating System (OS) and SAS versions, has given me a great insight into programming environment factors that I may otherwise not have been aware of.

Having a deeper understanding of your programming environment, for any language, can be an extremely useful and important thing. Having this knowledge can allow us to debug issues we see faster, and more importantly can ensure that code or macros we write are as portable across systems as possible.

If I wrote down everything here this would be a VERY long paper, therefore I have focused on some considerations of SAS Product versions, SAS system options, and ways that the OS can impact on our code that I see most often raising questions.

SAS PRODUCT VERSIONS

The first example is from the following SAS code which was being run by two programmers in seemingly similar environments:

```
proc power;
  twosamplefreq test = FM_RR
  ...;
run;
```

NOTE: The `...` in the code above is for simplicity only, other parameters were correctly specified.

For one programmer the above code was running fine, but the second programmer was having issues and SAS was giving them the following issues in the log:

```
WARNING 1-322: Assuming the symbol FM was misspelled as FM_RR.
ERROR: The NULLRELATIVERISK option is invalid for TEST=FM.
ERROR: The GROUPPROPORTIONS and NULLRELATIVERISK options cannot be used
simultaneously.
```

The two programmers had conferred and confirmed that they were both using PC SAS v9.4 on virtual Windows desktops. It was confirmed that for **`proc power`** the value of FM_RR was indeed valid for TEST= and everything else was filled out correctly.

So why the issue? The answer for this could be found by running the below code:

```
proc product_status;
run;
```

This procedure produces a list of which sub-products or packages are installed in the current SAS environment and although it confirmed that SAS v9.4 was indeed being run on both virtual machines, the procedure **proc power** is part of the SAS/STAT sub-product, and the versions of this package between the two sessions were noticeably different.

The initial programmer was running an install of SAS that had SAS/STAT version 14.3 and this ran the above code as expected, as confirmed by the SAS/STAT 14.3 documentation [1].

However, the second programmer that was encountering the issue was running SAS/STAT version 13.2. This older version of SAS/STAT [2] does not have FM_RR as a valid value of test=, which is why SAS was assuming FM_RR was instead a misspelling of FM.

The version of BASE SAS and which packages are available, which you can find using **proc setinit**, should always be considered when programming. However, if you are using functions or procedures that are part of one of the sub-products or packages, like statistical procedures in this case, it is worth checking which version(s) you are using and ensure that you are checking the correct reference documents when searching documentation.

The same can be said for other sub-packages which include SAS/GRAPH, SAS/AF and SAS/ACCESS Interface for PC Files among many more.

SAS OPTIONS

For the second example I wanted to look at SAS options. Although SAS has defaults, SAS options are commonly changed in the related config or autoexec files so may differ between setups. Most options can also be changed at any time during a SAS Session and this can lead to unintended consequences if the states of these are not known or controlled.

An easy example to illustrate the impact of options is OPTIONS VALIDVARNAME – this option states the type of SAS variable names that can be used or created during a SAS session. Details and valid values for this option can be found (for SAS v9.4) in SAS documentation [3]:

OPTION VALIDVARNAME

I will focus on just two of the extreme values for this option.

The most lenient option for SAS variable naming is using `options validvarname=any`. Variables can begin with or contain any characters (including blanks!) with restrictions only on the length of the variable name. SAS stores and writes the column name in the same case that is used in the first reference to the column.

At the other end we have `options validvarname=upcase` which is a lot more restrictive with names limited to: letters of the Latin alphabet, an underscore or numeral; must not start with a numeral; cannot contain blanks/spaces; cannot contain special characters except underscore; SAS stores and writes the column name in uppercase.

SPACES IN VARIABLE NAMES

The first obvious difference is the possibility of having a space in your variable name with `validvarname=any`.

The following will work with `validvarname=any` but error with `validvarname=upcase`.

```
data test1;
  'a var name'n = 200;
run;
```

This may seem fine for most programming, you just normally wouldn't create any variables names with spaces in, right?

But the impact of this option can become more evident if reading in external files where you have let SAS read-in the column names, for example from Column headers in an Excel file. Depending on the value of `validvarname` SAS may allow spaces, or replace these with underscores so you can end up with different variable names depending on this option.

It also means variables with spaces may have been created which can be confusing for newer programmers. To reference these variables use a name literal (surround the name in quotes and immediately follow with an n) as shown in the example above. For more information on literals, see SAS Help documentation [4].

So if reading in external files, you do need to be careful of this option.

CASING

With either option enabled, SAS will still process all column names ignoring case, so 'abc' and 'ABC' are considered the same variable during normal SAS programming regardless of this option. So often, the casing doesn't hugely matter when writing your code referencing variables.

So why do we care? It's not the processing we need to worry about this time, it is the way SAS stores and writes the column name which is different.

The value of the column name (NAME) in DICTIONARY.COLUMNS or SASHELP.VCOLUMN may depend on the `validvarname` option selected. Therefore, if you are looking up the variable from the above reference datasets, casing may matter as these are stored as case-sensitive strings, even though it does not impact the rest of your code referencing the variable.

Additionally the output of some procedures such as **`proc transpose`** and some SAS/STATs procedures that can turn column names into values (e.g. the value of _NAME_) or vice versa may also change. So if you write a SAS program using `validvarname=any` and create IF statements for _NAME_=, then those same lines may no longer work if the `validvarname` is changed to UPCASE, or vice versa. If transposing to create new variable these may end up with or without spaces in them, which impacts future code that may reference them and/or again impact usability if `validvarname` is changed.

SUGGESTION

Writing the code more defensively with knowledge of how various options can impact your code can help programs be made transportable, for example using the `upcase()` function when checking _NAME_. However, I often find it is easiest to use `validvarname=upcase` as default and specified at the start of programs I write, that way if they are run on a different SAS setup I know my **`proc transpose`** and other functions will still work as intended.

Just remember, if you change options during your code it is good practice to reset those options at the end of your program to avoid this affecting other programs you may run in the same session that may not have expected an option to change from the default. See PHUSE EU Connect 2022 Paper CT10 - Standardised SAS Macro Considerations for Submission-Ready Code [5].

EXTERNAL FILES – TERMINATING STRINGS

One of the more obvious differences in environment can be the OS on which you are working. Most of the time SAS defaults are set to deal with the OS in which it is running, however things can get confused if you are working in one OS with SAS servers running your programs on a different OS, e.g. using SAS Enterprise Guide in a Microsoft® Windows Environment to run SAS programs on Linux servers, or receiving files from a different OS.

It is particularly the latter situation of receiving and reading in files that has been the cause of many questions I've had over the years and there is often one common theme – the "terminating string".

THE TERMINATING STRING

Different operating systems use different "invisible" characters to signal that there is a new line/entry and thus often ending the current record when reading these files in.

By default, Microsoft Windows uses a combination of 2 characters: Carriage Return (CR) directly followed by Line Feed (LF). Unix based systems, including Linux, instead use just LF although sometimes a combination of CR and LF are acceptable as well. Classic Mac OS uses just CR and other systems can be different again.

As much as we should never rely solely on Wikipedia (although there is a large group of volunteers devoted to keep it accurate), this does help to show the basics, including the underlying Hex/Dec and Escape Sequence alternatives for these important non-printable characters [6].

SO WHY DOES THIS MATTER?

If you take a file created in Linux and open, or attempt to read it into SAS, in a Windows environment you will often see that the entire file looks as if it is on a single line, because it never finds a CR character directly followed by LF which it is looking for by default. Have you ever taken a file (e.g. even a SAS program) created or written in Linux and tried to open in Windows NotePad? This gives us the same thing, the entire contents will appear in a single long line. (Note that WordPad and many other editors somewhat account for different OS sources and terminating strings, so those often correctly display files.)

Conversely, opening a Windows file in Linux you will often see that your file has seemingly read in correctly, but sometimes the last variable is 1 character longer than it should be or has been read in as character rather than numeric. This is because it has read in the CR character as if it is any other non-important character in that last variable and this character is not valid for a numeric variable, so it must be character. This can be highlighted further if you let SAS read in the column names for you, SAS can sometimes default the last column name to "VARIABLEn", where n is the number of the last column (but this depends on your `validvarname` option). It has done this as a CR character is not a valid character under some `validvarname` values and therefore it couldn't fully read in the variable name and so SAS creates one for you.

OK, SO I'VE MENTIONED THE POTENTIAL ISSUES, BUT HOW CAN WE DEAL WITH THEM?

First, as ever, try to find out what OS/environment the file was created in, then take a look at the `termstr` parameter for either the `infile` or `filename` statements associated with that OS. The `termstr` ("Terminating String") parameter allows you to specify the expected new line characters of the file you are reading in which is great when it doesn't match the default of your current OS, or if you wish to make your code more flexible across different systems but know the file generated always comes in the same way.

Here is a quick `infile` example using `termstr`:

```
data test;
  infile myfile dlm=',' termstr=CRLF;
  input …;
run;
```

If reading in Windows files, use `termstr=CRLF`, for Linux files generally use `termstr=LF`, for others… well I think you get the idea. If in doubt there is no harm in trial and error! Or there are scripts that will convert between the two, or at least confirm the line endings. For the list of `termstr` valid values, or for more information on what it is, look at the `termstr` parameter for `infile` in the SAS documentation [7].

There are also OS specific versions of the documentation for Windows [8], Unix/Linux [9] and z/OS [10]. Note that the "default" of `termstr` changes between each OS to the default of the respective OS environment.

Note also that the same is equally true if outputting files, you can use `termstr` on `file` statements too.

REFERENCING FILES IN DIFFERENT OPERATING SYSTEMS

There are also other considerations when dealing with referencing external files in SAS depending on your Operating System.

*CASE SENSITIVITY (PARTICULARLY *.SAS7BDAT FILES)*

When referencing files one of the first things we must remember is that Unix based systems are case sensitive. You can, for example, have a file called "test.txt" which is completely different to a file called "TEST.TXT" or "TeST.txt" all existing in the same folder and having completely different contents and metadata. However, Microsoft® Windows by default is generally case insensitive when it comes to filenames, so the above filenames would all reference the same file in a folder and the casing used when referencing the file does not matter.

Most of the time this is self-explanatory, always be careful of filename casing in Linux when referencing files and folders (and ideally in Windows too, to avoid compatibility issues if your code and files are moved to another system).

However, another implication of case sensitivity in Unix means that one interesting thing can happen when SAS dataset files (*.sas7bdat) are sent from another vendor that have used uppercase letters in the filename (often due to a copy/move operation which may rename the file to uppercase). SAS in Unix based systems cannot read *.sas7bdat files that have uppercase letters in the filename [11].

The file may still appear visible in a library when mapped, but when trying to open or read it SAS will give you errors and searching the dictionary/SASHELP datasets for the file will normally find nothing. If you notice such behaviour while working in Unix/Linux or similarly case sensitive environments then take a look at the filename of the *.sas7bdat and see if it has any uppercase letters.

FOLDER SEPARATOR

Another difference is the delimiter character used between folders in a path which changes between systems.

Traditionally Windows default is to use a backslash \ in a path to separate folders and subfolders on a mapped drive, while Unix based systems use forward slash / instead, other systems can use a dot . or colon : or allow a mixture. That said, more modern versions of Windows tend to be more lenient by recognising both \ and / as path delimiters so folders can still be referenced using the wrong folder delimiter, while Unix remains less forgiving.

Remembering which OS you are dealing with is useful, but equally you can program in steps to check which OS is in use and then assign a macro variable to the delimiter, e.g. something like:

```
%if %sysfunc(find(&sysscp,LIN)) %then %let fd=/;
%else %if &sysscp=WIN %then %let fd=\;
%else %put %str(ERR)OR: Unknown Operating System: &=sysscp &=sysscpl. Please
check Folder Path delimiter;
```

You can then use this folder delimiter (&fd.) when needed, rather than writing / or \ as folder separators.

UNIVERSAL NAMING CONVENTION (UNC)

Also, keep in mind that UNC paths are not supported in Linux, only Windows. Files located on servers you may have previously accessed using paths in SAS like `\\myserver.net\folder` will no longer work if the program is moved to a Linux environment.

CONCLUSION

This is just a glimpse of the various ways that SAS packages, SAS system options, and the OS environment you are using and/or receiving files from can impact the programs that we write.

The associated presentation given as part of the PHUSE EU Connect 2023 shows real-life scenarios of where these considerations impacted programming activities on studies. The more you understand about your environment and differences that can occur, the faster you can debug, or proactively code to avoid running into these issues.

Further considerations that I have not been able to cover in this paper include, but are not limited to:

- SAS Version
  - Additionally noting any maintenance release version, e.g. basic macro `%if` statements are executable in open code (outside a macro) in SAS v9.4 but only from Maintenance Release 5 onwards
- More considerations of the OS specific to coding, e.g.
  - Availability of options, such as FILELOCKS
  - Which commands are available in X Command (if enabled)
- Session and file character encoding
- Interactive vs. batch running
- SAS State, i.e. Locked-Down or not
  - Availability of X Command
  - X Command will also determine if procedures like PROC GROOVY are available
- Differences between the various SAS editors and place of execution, e.g.
  - Running code on servers vs. local
  - Use of Display Manager, WINDOW etc.
  - Default open ODS destination(s)

With all that said, I'll leave you with some final reminders:

- Your SAS version goes beyond the BASE SAS version. Learn what packages are installed and their respective versions.
- Different systems will have different defaults.
- To make code transportable, ideally program to the most restrictive (within reason…).
- The option `validvarname` has a wider reach than just variable naming. When those names are used to create values (like in a transpose) other considerations come into play.
- Consider which OS you are using, and which OS may have been used to create external files. If in doubt and you run into issues, have a play with the various options! NO NEED TO CHANGE THE INPUT FILE.
- If in doubt, use lowercase filenames for everything (also this aligns with some regulatory requirements anyway).
- Some of the considerations noted here, although written primarily with SAS in mind, will impact other programming languages too.
- **Know your environment!**

REFERENCES

[1] The POWER Procedure, SAS/STAT version 14.3 documentation, SAS®
https://documentation.sas.com/?docsetId=statug&docsetTarget=statug_power_syntax83.htm&docsetVersion=14.3&locale=en#statug.power.powtfrqtest

[2] The POWER Procedure, SAS/STAT version 13.2 documentation, SAS®
https://support.sas.com/documentation/cdl/en/statug/67523/HTML/default/viewer.htm#statug_power_syntax68.htm#statug.power.powtfrqtest

[3] VALIDVARNAME= System Option, SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation, SAS®
https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lesysoptsref/p124dqdk8zoqu3n1r4nsfqu5vx52.htm

[4] SAS Constants in Expressions, SAS® 9.4 Language Reference: Concepts, Sixth Edition, SAS®
https://documentation.sas.com/doc/en/lrcon/9.4/p0cq7f0icfjr8vn19vyunwmmsl7m.htm

[5] CT10: Standardised SAS Macro Considerations for Submission-Ready Code, PHUSE EU Connect 2022, Will Greenway
https://phuse.s3.eu-central-1.amazonaws.com/Archive/2022/Connect/EU/Belfast/PAP_CT10.pdf
https://phuse.s3.eu-central-1.amazonaws.com/Archive/2022/Connect/EU/Belfast/PRE_CT10.pdf

[6] Newline, Wikipedia®
https://en.wikipedia.org/wiki/Newline

[7] INFILE Statement, SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation, SAS®
https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lestmtsref/n1rill4udj0tfun1fvce3j401plo.htm

[8] INFILE Statement: Windows, SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation, SAS®
https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/hostwin/chifoptfmain.htm

[9] INFILE Statement: UNIX, SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation, SAS®
https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/hostunx/p18r7xw5gg6is3n127xdj69d6b4p.htm

[10] INFILE Statement: z/OS, SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation, SAS®
https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/hosto390/p06y69omp86scmn1i9nvllynndga.htm

[11] Usage Note 15989: In UNIX environments, the names of SAS® data sets and external files must contain only lowercase characters, SUPPORT / SAMPLES & SAS NOTES, SAS®
https://support.sas.com/kb/15/989.html

ACKNOWLEDGEMENTS

I would like to acknowledge and thank Elizabeth (Liz) Meeson, and Tony Cooper for their review of this paper and the associated presentation.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Will Greenway
Plus-Project Partnership
Suite L3CA, No1,
Booths Park,
Chelford Rd,
Knutsford,
Cheshire,
WA16 8GS

Email: will.greenway@plus-project.co.uk
LinkedIn: https://www.linkedin.com/in/willgreenway/
Web: https://www.plus-project.co.uk/