Paper 3062-2019

Maxims of Maximally Efficient SAS Programmers

Kurt Bremser, Allianz Technology

ABSTRACT

The Maxims are a set of simple rules intended to improve the life of a SAS programmer. They provide best practices, hints, cautions, stuff for thinking, and even some humor.

INTRODUCTION

While helping a poster on the SAS Communities, I got repeated questions like “but what happens if …”. I finally started to post an answer that went “just try it”, and realized this would be a good rule-of-thumb for any SAS programmer. So I jokingly called it “Maxim 4 of the Maxims of Maximally Efficient SAS Programmers”, with a not-so-veiled reference to the “70 Maxims of Maximally Effective Mercenaries” from schlockmercenary.com, a web comic known quite well among science fiction fans (of which I am one). Almost immediately I was asked where the rest was, and so I wrote an article containing my first maxims. Since it received quite a lot of positive feedback, I kept on adding and improving, and right now there are 50 maxims all-in-all.

THE MAXIMS

Maxim 1: Read the Documentation.

SAS provides extremely well done documentation for their products at documentation.sas.com.
For programming issues, go to SAS 9.4 and SAS Viya 3.4 Programming Documentation.
“Read the Documentation” also implies “learn to read the documentation”; only frequent use of the website will enhance your navigating skills there.
Once you know your way around the documentation, you will observe an improvement in your problem-solving skills by orders of magnitude.

Maxim 2: Read the Log.

Everything you need to know about your program is in the log. Interpreting messages and NOTEs is essential in finding errors.

Maxim 3: Know Your Data.

Having a clear picture of data structures – variable types, lengths, formats – and content will provide you with a fast-path to solving problems. Many simple problems can be cleared by taking a look at the "Columns" section in dataset properties. Use proc contents frequently.
Know physical aspects of your datasets; size, location and type of storage have a profound effect on performance of the steps dealing with your data.

Maxim 4: If in Doubt, Do a Test Run and Look at the Results. If Puzzled, Inquire Further.

SAS is in its core an interpreting language. Running one step is just a few mouse-clicks away. Use that to your advantage.

"Try it." (Grace Hopper, Admiral, US Navy)

One caveat: do not overwrite datasets in the same step. This will let you test individual steps repeatedly without the need to recreate the input in case of failure.

Maxim 5: Ask, and You Will Be Answered.

SAS Technical Support and the SAS user community stand at the ready to help you. Provide a clear question, example data, your code (with or within the log, see Maxim 2: Read the Log.), and where you seem to have failed, or where the result does not meet your expectations. Help will be on the way.

Maxim 6: Google Is Your Friend.

Just entering the text of an error message or problematic issue, prepended with “SAS”, will often yield resources with at least a hint for solving your issue on the first result page; the same is true for other search engines.

Maxim 7: There Is a Procedure for It.

(The exception proves the rule)
Learn to use prefabricated procedures for solving your tasks. 5 lines of proc means may equal 20 lines (or more) of data step logic.

Maxim 8: There Is a Format for It.

(The exception proves the rule)
As with Maxim 7, SAS provides a plethora of formats for input and output. Use them to your advantage. If one that fits your needs is not present, rolling your own will usually beat complicated data step logic. Also see Maxim 21: Formats Beat Joins.

Maxim 9: There Is a Function for It.

As of 2019-01-31, documentation.sas.com lists 649 data step functions and call routines. It is almost impossible to not find a function that makes solving an issue easier, or solve it altogether.

Maxim 10: SQL May Eat Your WORK and Your Time.

With large datasets, the way PROC SQL handles joins will lead to the buildup of large utility files, with lots of random accesses to these. This may well prove much less efficient than a join done with a data step and the necessary proc sort steps.

Maxim 11: A Macro Is Not Needed.

(The exception proves the rule)
There may be no need for repeating code when using by-group processing can do the trick. The macro language is also not meant for handling data - like calculating dates - but for creating dynamic code. Instead of creating lists in macro variables, store them in datasets and use call execute from there. Do calculations in data steps and save the results in macro variables (for later use) with call symput().

A macro cannot solve anything that can't be solved with Base SAS code. But it can reduce the effort needed to write that code, and it can help in making code data-driven. Also see Maxim 33 in reference to call execute.

Maxim 12: Make It Look Nice.

Proper visual formatting makes for better code. Use indentation to make semantic blocks visible. What is on the same logical level, needs to be at the same column.

Avoid overlong lines, make a block of lines if necessary (that 80 character limit of old is not so bad at all).

Create a consistent coding style and use it; the next one to maintain that piece of code might be your five-year-older self. Be nice to her.

If a larger group of programmers work together, agree on a common coding style, so everybody can “feel at home” in shared code.

Make frequent use of comments.

Today, use of lowercase in code is preferred by most coders. It is more pleasing to the eye and allows the use of uppercase for emphasis.

Maxim 13: When You’re Through Learning, You’re Through.

Quote from (among others) Will Rogers, John Wooden

As long as you keep your ability and will to learn, you are alive. When you stop learning, you may not be dead, but you start smelling funny. Never say "I don't have the time to learn that now". The time to learn is NOW.

“Much have I learned from my teachers, more from my colleagues, but most from my students.” - from the Talmud.

Maxim 14: Use the Right Tool

“How many times do I have to tell you, the right tool for the right job!”
(Montgomery Scott, Captain, Starfleet)

Never restrict yourself by a simple “Use XYZ” or “Don't use XYZ”. If something is better solved with a certain procedure, use it. If you don't yet know it, learn it (see Maxim 13). If a 3rd-party tool is better suited, use it (think of DBMS operations; have them done in the DB itself). Leave operating system tasks to the operating system (see Maxim 15).

Maxim 15: Know Your Playing Field.

Make sure to gain knowledge about the environment in which SAS is implemented. Know the system's layout, its basic syntax, and its most important utilities. Especially UNIX is rich with tools that can and will make your life easier. Control the system, don't let the system control you.

Maxim 16: If It Isn’t Written, It Is Not.

Anything that is not properly documented, does not exist. It may look like it was there at some time, but the moment you have to deal with undocumented things (programs, data, processes), it is as if you are starting from scratch. The Jedi programmer spends 90% of her time documenting, and reading documentation.

“I should have written that down. - Dilbert”
(Scott Adams)

Maxim 17: Do It the SAS Way.

A data warehouse needs different structures than databases, and it needs different structures than spreadsheets. Adapt your thinking away from normalized tables (redundancy is avoided in DBMS designs, but is typical in data warehousing tables used for analysis), and also away from cells (spreadsheets have no mandatory attributes for columns, SAS tables have). See a SAS table as just the technical entity it is, and model it according to the analytic need.

Maxim 18: Separate Your Names.

Do not use names of predefined SAS functions, call routines or formats for your objects (variables, datasets). This avoids confusion and prevents hard-to-trace errors when a simple typo happens.

Maxim 19: Long Beats Wide.

(Don’t keep data in structure)
In the world of spreadsheets, people tend to line up data side-by-side, and put data items (dates, categories, …) into column headers. This runs counter to all the methods available in SAS for group processing, and makes programming difficult, as one has variable column names and has to resort to creating dynamic code (with macros and/or call execute) where such is not necessary at all if categories were represented in their own column and data aligned vertically.

Example:

ID	Salary_Jan	Salary_Feb	Salary_Mar
1	100	110	105
2	195	185	190

Table1: Wide format dataset

should be

ID	Month	Salary
1	1	100
1	2	110
1	3	105
2	1	195
2	2	185
2	3	190

Table2: Long format dataset

There are times where a wide format is needed, eg when preparing data for regression analysis. But for the processing and storing of data, long formats are always to be preferred.
Dynamic variable names force unnecessary dynamic code.

Maxim 20: Keep Your Names Short.

Short, concise variable names make for quicker typing with less risk of typos, and make the code more readable. On the other hand, keep your variable names meaningful.
Put additional information in labels, where it belongs.

Examples:

```
employee_average_salary_for_a_year
```
(bad)
```
'This is the avg. annual salary'n
```
(even worse)
```
aasal
```
(bad)
```
avg_annual_salary
```

(good)

Maxim 21: Formats Beat Joins.

Creating a format from a dataset lets you do a “join” on another dataset in one sequential pass. As long as the format fits into memory, it provides an easily understandable means to avoid the sort of a large dataset.

A more advanced method to avoid the classical sort & merge or SQL join is covered in Maxim 50: Check Your OBS= System Option.

Often OBS= was set to limit the number of observations for tests. This will cause seemingly inexplicable terminations of steps later on.

Maxim 51: Hash Can Fire You Up.

Maxim 22: Force Correct Data.

When reading from external sources, don't rely on proc import. Instead use a data step that will fail with an error if an unexpected event in the input data happens. This lets you detect errors early in your analytic chain. Maxim 22 specifically precludes using the Excel file format for data interchange, because of the many involved automatisms.

Maxim 23: Recursion.

See Maxim 23: Recursion.

Maxim 24: Return With a Zero.

No batch program can be allowed to end a successful run with a non-zero return code. No WARNINGs, no ERRORs.

Write your code so that programs stay “clean” for a run considered successful.

Any unexpected event, OTOH, shall cause a non-zero return code, so a chain of jobs is halted at the earliest possible moment.

Maxim 25: Have a Clean Log.

The log of a production-level process should be free of any extraneous NOTEs. Automatic type conversions, missing values and so on must not be tolerated.

This allows for easier detection of semantic problems or incorrect data, as these will often cause unexpected NOTEs.

As long as a part of your log is "unclean", all following ERRORs or other problems might be caused from that, and most of the time are not worthy of your attention. Debug from the top down.

Maxim 26: .sas Trumps Catalogs.

While it is possible to keep code in catalogs, storing it in simple text files is better. SAS catalogs are specific to SAS versions and operating system environments. Text files are never version or operating system specific, eventual changes in codepages are handled by SAS itself over the IOM bridge or by file transfer utilities.

Text files also enable you to use external tools for text handling (think grep, awk, sed and so on), and versioning systems.

In the same vein, always store the code of proc format (and its related cntlin datasets), and do not rely on the created formats in their catalogs.

Maxim 27: Textuality Rules.

Use plain text whenever possible. Use simple file formats for data transfer, so you can inspect the data with a text editor. Prefer modern, XML- or JSON-based files (even when compressed, like xlsx) over older, binary formats. Store your code in .sas files, use Enterprise Guide projects as temporary containers only while developing. Text lends itself well to versioning and other tools that help the programmer, binary files don't. See Maxim 26.

Maxim 28: Macro Variables Need No Formats.

Keep in mind that '01jan1960'd and 0 represent the same value in SAS code. Therefore you only need formatting of macro variables when they are meant for display (eg in a title statement). When you only need them for conditions or calculations, the raw values are sufficient and easier to create and use.

Maxim 29: When In Doubt, Use Brute Force.

(Ken Thompson, creator of UNIX)
Use simple, easy-to-understand code. Only revert to “clever” algorithms when circumstances require it. This is also a variant of the KISS (**K**eep **I**t **S**imple, **S**tupid) principle.

Maxim 30: Analyze, Then Optimize.

Do not engage in optimizing before you know where the bottleneck is, or if there is a bottleneck at all. No need to speed up a step that takes 1 second in real life. Keep your code simple as long as you can (see Maxim 29).

“Premature optimization is the root of all evil”
(prob. C. A. R. Hoare)

Maxim 31: Computers Are Dumb.

Never trust a piece of software to do on its own what you intend it to do. If you need a certain order, sort or use “order by” in SQL. If a variable should be of certain length, set it. Do not rely on the guessing of proc import, roll your own data step. (see Maxim 22: Force Correct Data.)

Even if things work in a certain way on their own *now*, this does not guarantee anything for the future. New software releases or changes in data might wreck your process. Take control.

Maxim 32: No Test Without COMPARE.

When making a change on an existing piece of code, never release it without running a regression test against data from the previous version with PROC COMPARE. This makes sure that only the changes you wanted (or in the case of an optimization, no changes at all) are introduced.
Even use COMPARE at crucial steps in your development process, so you catch mistakes when they are introduced, instead of having to search for that spot later.

If COMPARE takes its time, enjoy your coffee. It's time well spent.

Maxim 33: Intelligent Data Makes for Intelligent Programs.

The Jedi programmer strives for intelligent data structures. The code will then be intelligent and well-designed (almost) on its own.

“Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”
(Linus Torvalds)

When dealing with time-related data, use SAS date and datetime values.

Categories are best kept in *character* variables.

Automate the creation of formats (eg from domain-tables in the DB system). Use these formats instead of select() blocks or if-then-else constructs. See Maxim 8: There Is a Format for It.

Create control datasets and use call execute() to create code dynamically from them.

Keep (especially non-ASCII) literals out of the code (by putting them in data); this will avoid problems when code is moved across systems.

Use properly structured formats for dates when the sequence matters (ISO-norm YYMMDD instead of MMDDYY).

Also see Maxim 19: Long Beats Wide.

Maxim 34: Work in Steps.

Don't solve a task in one everything-including-the-kitchen-sink step. Create intermediate datasets/results, so that each step solves one issue at a time that can easily be comprehended. See Maxim 29: When In Doubt, Use Brute Force.

In the same vein, don't implement multiple changes to a program in one iteration. Instead do a check-in for every single logical change. This allows regression tests for every change, and if something breaks which was not immediately obvious in regression testing, it is easier to step back and find which change introduced the bug.

Maxim 35: There’s Two Kinds of Stupidity.

1. “This is old, and therefore it's good”.
2. “This is new, and therefore it's better”.

Fads have come and gone, but every now and then there's a gem in all the rubble. Be critical, but open-minded.

Maxim 36: No Work Is Done Until the Final Test Has Run.

The finishing step of every piece of work must include a test of the program in exactly the same environment it will be used (same server, same user, same batch script, etc). Only then can you be sure that the first use does not result in a call for support that interrupts your well-earned coffee break. “Normal” users may have a more restricted environment than developers, so keep that in mind when developing code for them.

Maxim 37: Perfection.

“Perfection is attained not when there is nothing more to add, but when there is nothing more to remove.”
(Antoine de Saint-Exupery)

Another application of the KISS principle; also think of the practice of “Muntzing” applied to programming.
Get rid of unnecessary code, redundant variables, irrelevant observations.

“Perfection is not attainable, but if we chase perfection we can catch excellence.”
(Vince Lombardi)

Maxim 38: Beware of the Dread God Finagle.

And heed the words of his mad prophet Murphy:
“What can go wrong, will go wrong”.

Write your code in this light, and include safeguards in expectation of the unexpected.

Make sure that you are notified when the unexpected happens (eg make a batch program crash on invalid data, so the scheduler catches it).

(With regards to Larry Niven.)

Maxim 39: Be Human.

Everybody makes mistakes, including you. Be kind when dealing out critique, as someone might be forced to critique you 5 minutes later.

There's always a better, and a worse. You're not the worst, but surely you're not the best. Learn from both, and be kind to both.

And remember that the code is not the coder. Very nice people can write incredibly ugly code, and vice versa.

Maxim 40: Talk to the Customer.

Having a good line of communication with the ultimate consumer of data is essential.

Make them formulate their needs in clear, unambiguous language; often this act of questioning themselves provides a major advance in solving the problem.

Never solve a problem that you *think* the customer needs solved.

Deliver example results early and often.

Maxim 41: Wisdom Emerges from Experience.

Unfortunately, most experience emerges from stupidity.

Maxim 42: It's Not the Answer, It's the Question.

A well-formulated question may already lead you in the right direction before you speak it out loud or write it down.

An ill-formulated question will get you the equivalent of “42”.

Maxim 43: Keep the Final Goal in Mind.

Which information needs to end up where? The answer to this question shall guide your selection of tools and methods, not the other way round. Information is at the core of your work, and delivering it correctly, timely, and usable is your ultimate goal.

Once the content is delivered and accepted, you can think of “prettying up”.

All this implies that the requirements have been refined and clearly worded (see Maxim 42).

Have test cases defined against which you can prove your code.

Maxim 44: Leave No Spaces.

All operating systems and programming languages use blank space as the prime separator of names and objects. It is therefore a VERY BAD idea to have spaces in names (of files, directories, columns and so on). Even when a certain environment provides a means to do so (encapsulating file names in quotes in UNIX and Windows, or the 'my name'n construct in SAS), one should not use that. Underlines can convey the same meaning and make handling of such names easier by orders of magnitude.

(The fact that certain products make happy use of blanks in file- and pathnames does not make this practice right; it just illustrates the incompetence accumulated in some places.)

Maxim 45: TANSTAAFL.

There **A**in't **N**o **S**uch **T**hing **A**s **A** **F**ree **L**unch.
A big salute to the great Robert A. Heinlein for this perennial truth.

Good, durable code requires thought and hard work. Be prepared to provide both.

Everything of value comes at a price, although the price is often not measurable in money.

Also see
"Fast. Cheap. Good. Select any two."

Maxim 46: Beware of the Hidden Blanks.

One tends to forget the fact that character variables have a fixed defined length, and values will always be padded with blanks up to that length. So it is always a good idea to use trim() when doing comparisons or concatenations.

Example:

```
   if charvar in ('AAA','BBB','CCC');
```

will work even when charvar is defined with a length > 3, but:

```
   if findw('AAA BBB CCC',charvar) > 0;
```

will fail if charvar has a length of 5 and will therefore contain '`AAA  `', which can't be found in the comparison string.

Similarly:

```
   charvar = charvar !! 'some_other_string';
```

will invariably surprise the unwary novice (and even the seasoned hand every now and then).

Maxim 47: Set a Length.

When creating new character variables, always set a sufficient length explicitly. When character variables are assigned literals, the first such assignment sets the length, causing confusion afterwards, because longer data will be truncated.

Example:

```
   flag = 'no';
   if (condition) then flag = 'yes';
   if flag = 'yes' then …;
```

Surprise!

Maxim 48: The Dot, the Dot, Always the Dot.

Make it a habit to always terminate macro variable references with a dot. It's never wrong to have the dot, but omitting it can have consequences and cause confusion.

See:

```
libname myexcel xlsx "/path/excelfile&year.xlsx";
```

Maxim 49: Don't Forget to run;

People frequently omit the `run;` statement, because SAS takes care of step boundaries all by itself.

But there are circumstances when this can cause a hard-to-detect problem; imagine this part of a macro:

```
data _null_;
call symputx('nobs',nobs);
set have nobs=nobs;
%do i = 1 %to &nobs;
```

Since no `run;` was encountered, the data step was not yet compiled and executed, leading to a "symbolic reference not resolved" error.

Bottom line: as with dots in macro variable references, the `run;` never hurts, but its absence sometimes does.

Maxim 50: Check Your OBS= System Option.

Often OBS= was set to limit the number of observations for tests. This will cause seemingly inexplicable terminations of steps later on.

Maxim 51: Hash Can Fire You Up.

The rather recently (version 9) introduced hash object provides a high-performance tool for in-memory lookups that helps solve complicated issues which required complex programming earlier on, or multi-step operations consuming time and disk space.

Learning its basic operation, and use in advanced situations is a must for the aspiring SAS programmer.

A very good introduction is found in SAS® Hash Object Programming Made Easy by Michele M. Burlew.

Maxim 52: Take a Break.

Whenever you run into a seemingly unsolvable problem, turn away from it for a while. Play a game (or watch one), listen to music, have a conversation with your S.O., make up a good meal, or just have a good coffee. Or have a night's sleep.

Cleansing your mind from all the (non-helping) thoughts you already had will open it up for the single new one you need to get over the obstacle.

It may even be the case that a solution comes to you in your dreams (I had that happen to me once; got up in the middle of the night, took some notes, and solved the issue in just a few minutes after arriving back at the office).

CONCLUSION

It's in the hope of the author that SAS programmers/coders, beginners and advanced, may profit from the Maxims. They should give food for thought, provide a basis for discussion, and invite critique.

REFERENCES

The online version of the Maxims can be found at https://communities.sas.com/t5/SAS-Communities-Library/Maxims-of-Maximally-Efficient-SAS-Programmers/ta-p/352068.

ACKNOWLEDGMENTS

For encouragement, critique, and suggestions I want to thank:

ChrisBrooks
BeverlyBrown
RW9
ballardw
SASKiwi
Doc_Duke
mftuchman
bentleyj1

who have all contributed to the Maxims, one way or another,
and to Reeza, the grandmaster of problem solvers.

A big "thank you" goes to Sandra and Howard Tayler, who let me paraphrase their "Maxims".

RECOMMENDED READING

For fun and wasting time, visit
https://www.schlockmercenary.com/

CONTACT INFORMATION

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

Kurt Bremser
Allianz Technology Austria
kurt.bremser@allianz.at
https://communities.sas.com/t5/user/viewprofilepage/user-id/11562