SAS 8.x/9.x Coding Standard

Yongling Ding, Ph.D, CFA, FRM

January 6, 2004

Contents

SAS is such a huge system that it is impossible to cover all of its features in a short note like this. This note focuses primarily on the language features available for programming, which include macro facilities, SAS/IML, and to a less extent SAS/AF. SAS has its roots in the mainframe system, and is built around a large collection of procedures with macros serving as a control mechanism. Compared to the traditional programming languages, data types play minor roles in SAS. Even more unfortunately, some SAS constructs are not good practice according to modern programming standards. But we cannot afford to discard all those procedures or functions, and thus have to work under those constraints. These idiosyncrasies require special attention.

As a proprietary software system, SAS is likely to evolve over time and some of the discussion may be relevant only for version 8.x/9.x. This note is written with Windows or Unix types of target platforms in mind. Some discussion may not be relevant for mainframe environments, especially the part regarding the file sytem.

1 Layout

A project workspace is a directory structure holding source codes as well as documentation, intermediate files, and utilities.

1.1 Package

A package is defined as the container for all the entities in a directory. A package may contain SAS programs or catalogs, but not both. A catalog serves as a subcontainer within a package for SAS/AF programs. Other files, including static data files that is an integral part of certain algorithms and scripts written in other programming languages, can also enter here.[1]

Unlike Java or C++, SAS does not have a namespace mechanism beyond library references. The name of a package directory is referred to as the package name. A package name should reflect the logical hierarchy of that package as if it were a Java package but with "`.`" replaced with "`_`". To shorten package names, only a part of the full name may be used. Usually, this includes a vendor identifier followed by a package identifier. All letters should be lowercase. SAS places a serious limitation on the length of entity names. Since functions will have their package name as a part of their names, it is important not to use lengthy names for packages. For the same reason as well as lack of benefits, packages should not be nested.

1.2 Files and Directories

As SAS dictates the meaning of file organization for catalogs and autocall libraries, there is very little freedom in organizing entities into files or directories for these entities. For non-macro SAS programs, no macro definition should be included, and each program should perform a relatively atomic operation. Autocall macros, instead of the `%INCLUDE` statement, should be used to call another piece of code.

The naming of files and directories should assume that their names are case insensitive.

1.3 Workspace

The name of a workspace root directory can be chosen freely because no reference should be made to this name and all references to subdirectories should be relative.

- `./` holds the build configuration script for the default platform. General information files such as `README` and `ChangeLog` should also be placed here. Build configuration refers to the setup of a build envrironment for a specific system with all the machine dependent information such as physical paths, host names, etc.

[1]The use of catalog is discouraged in development. Although SAS proprietary storage mechanism may have some performance benefits, it makes it rather difficult to view and edit programs and track versions outside the system. Since compilation takes place only once in runtime, the performan gain is likely to be modest in a production environment. When such a measure is indeed necessary for some other reasons such as guarding the source codes, the convertion to catalogs should become a part of build process.

- `bin/` exists only if there are precompiled executable programs that are distributed with the project. Programs that can be built from sources should not be left here.
- `config/` holds the auxiliary files for the build configuration.
- `doc/` holds the documentation for the project. Some of them may be generated automatically. Typical subdirectories are "`api/`" for API references, "`des/`" for design, and "`use/`" for usage. Alternatively, subdirectories may hold the same set of documentation in different formats.
- `etc/` holds the template configuration files and data files for runtime environments. Global runtime options may be defined in files under this directory. If that is the case, there should be a way to override the runtime options in each configured runtime environment without touching files in this directory.
- `examples/` holds all examples if any. Each sub-directory contains an example.
- `lib/` exists only if there are precompiled libraries or some external programs distributed with the project. Libraries that can be built from sources should not be left here.
- `port/` holds the files necessary for non-default build systems. These may include `Makefile`, scripts, or some proprietary project files. Each build system should have its own directory with the name being its identifier. However, intermediate build files should not enter any of the subdirectories.
- `maint/` holds scripts and other files used in the project management and maintenance.
- `src/` is the parent directory for packages. This directory may also contain files relevant to all packages.
- `test/` holds test cases as well as scripts used in testing.

The workspace does not contain any intermediate file generated during the build process. Those files as well as the files generated from the build configuration should be placed in a working directory.

Source control, when used, should not interfere with the above rules.

2 Naming Convention

Names should be meaningful in the problem domain. The SAS naming is case insensitive, but the convention here pretends that it is case sensitive. This makes the programs more readable. It is just that no two names can differ only in cases.

2.1 User-defined Types

User-defined types include class entries defined in a catalog. All user-defined type names should start with an uppercase letter. In a name that consists of more than one words, each word following the first one should start with an uppercase letter and all the other letters in that word must be in lower case, even if the word is an acronym. Each word in a type name should look like an adjective to the immediately following one.

2.2 References

The references for libraries should be one word starting with a capital letter.[2] There should be no need to name file references directly as the function `filename()` can be called to generate a unique reference for any file via a macro call. In rare cases where this becomes necessary, a special rule can be made.

2.3 Methods

The first letter of a method name should be in lower case.[3] Most method names should start with a verb. Verb "`get`" and "`set`" are reserved for accessors. Verb "`is`" is reserved for status checking. Verb "`has`" is reserved for property checking. Verb "`compute`" is reserved for lengthy computation, while verb "`calc`" is for simple calculation.

2.4 Functions

SAS does not have the function mechanism available in the traditional languages. The term "function" here refers to macros as well as IML modules.

All function names should start with the name of the package to which they belong followed by their concrete names. The names should all be lowercase with words separated by "`_`". The name of a nested IML module should start with the parent module name and end with a suffix "`_`". An extra suffix "_" is also needed for macros intended for internal use. This suffix distinguishes such macros or IML modules from other "public" functions even though SAS has no mechanism of enforcing such a protective mechanism.

2.5 Variables

Global macro variables should be defined for very specific purposes such as configuration. Normally, such variables should be treated as if they were constants

[2]It is important to use a shorter name for a library reference not only to stay within the naming limitation but to reduce typing and then instances of line breaks.

[3]Although both variables and methods start their names with a lowercase letter, this rule will not blur the difference between them. First, variables in an object scope have a prefix. Second, calls to a function have "()" immediately following the name. Third, the first word of a method name is more likely to be a verb, while the first word of most variable names is a noun.

outside the definition unit. Such faked constants should be named with all capital letters where words are separated with "`_`". Global variables defined for a common task should be grouped together as indicated by the same first words such as "`CONFIG_`".

No other type of global macro variables should be declared, and thus no corresponding rules are defined. For rare cases where global variables are absolutely needed, it is so special that a special rule can be made up just for that.

Each variable has a name component. Based on the variable type, its name may have a prefix or a suffix.

- Global symbols inside `PROC IML` should have prefix "`_`".
- Member variables of a class entity in a catalog should have suffix "`_`".
- Local macro variables or IML variables in a module cannot start or end with "`_`". Method and function arguments should be treated as local variables.
- The field names defined in data steps or nested SQL statements but not kept in any persistent storage should be followed by suffix "`_`".
- A name component should start with a lowercase letter. Most name components should start with a noun. This rule emphasizes the categorization of variables. It can be viewed as a generalization of cumbersome Hungarian naming convention popular in Windows programming. Instead of marking the actual data type of a variable at the beginning of its name in lower case, this rule asks for marking the variable type by a term in the problem domain. For example, "`dateStart`" is preferred to "`startDate`" if we want to emphasize "`date`" as a type, or "`startDate`" is preferred to "`dateStart`" if we instead want to emphasize "`start`" as a type.

Obvious loop variables are exempt from rules. They can take simple names like `i` or `j`.

2.6 Tables and Views

SAS data sets are a special type of SQL tables, while queries are a special type of SQL views. Both of them should follow the same rules defined here.[4] A catalog should be named as if it is a part of a package name, one word of all lowercase letters.

The name of a table or view consists of two parts: a prefix and a name. The prefix is either "`t_`" for tables or "`v_`" for views. A name component consists of words with each starting with a capital letter. Each table or view should have a two-letter acronym that shall become the prefix for all the fields in that table or view.

[4] The naming rules for tables and views apply only to the entities under control, and certainly cannot carry over to those created outside of the project.

The field name in a table or view consists of three parts: a table/view acronym, a type indicator, and a name component. The type indicators for the ANSI SQL-92 data types supported in SAS are listed in Table 2.6.[5] Other proprietary data types supported by certain vendors can be easily mapped into one of these type indicators. The name component should be formed with words where each starts with a capital letter.

Table 1: Type Indicators for SQL Data Types

Type Indicator	SQL Data Types
c	CHAR
v	VARCHAR
i	INTEGER, SMALLINT
f	FLOAT, REAL/DOUBLE PRECISION
n	NUMERIC, DECIMAL
d	DATE, TIME, TIMESTAMP

An exception to the above rules can be made for the fields strictly for internal use. Such fields are named like local variables with a suffix "`_`".

3 Coding Guidelines

It is important to follow the following guidelines in coding.

3.1 Functions and Methods

- Macros that do not span across procedure or data step boundaries, macros that expand to one or more data step and/or procedures, and IML module calls and definitions should not be mixed in a common package. This practice makes it easy to spot the style of a macro definition.
- Prefer keyword parameters to positional parameters in macro definition unless there is only one argument.
- All variables inside of a macro definition should be declared `%LOCAL` unless the purpose of a macro is to manipulate some global variables.
- When a function accepts a variable number of or different types of inputs, it is usually cleaner to delegate the actual work to several other macros defined in the same file that is otherwise inaccessible.[6]

[5]SAS supports only fixed-length characters and double-precision floating point numbers natively. The conversion between these two native data types and other SQL data types are handled by the drivers in SAS/Access.

[6]This uses the feature that all macros in an autocall macro file will be compiled even though only the primary macro will be executed directly. This eliminates the need to create another file to hold such a macro, but it does not hide this macro from global scope after the compilation.

- Do not nest macro definitions.
- Prefer function or subroutine calling convention to directly running data steps or procedures in a macro definition wherever possible. This results in maximum flexibility.
- Macro arguments that directly correspond to some well-known SAS options should have the same names as those options. The examples include "`OUT`", "`DATA`", and etc.
- IML module definitions should be wrapped in macros to allow flexible inclusion. This practice makes it easy to include IML modules in a IML procedure.[7]
- IML module calls should be wrapped in macro definitions to provide named parameters and default values if necessary.

3.2 Implementation

- No global macro variable is allowed unless required by some application framework. When global macro variables are deemed necessary, it is better to treat them as constants or have a centralized mechanism to manipulate them if updating is needed as well. An exception is the global symbols in an IML procedure, which may be used conveniently to create a weak closure with a module.
- Do not nest quoted macros. Where it is necessary to quote a quoted variable, that variable should be unquoted first with `%unquote()`.
- Data sets should usually be the driving force of a SAS application, and the integrity and consistency of data should be maintained separately from the main logic as much as possible.
- The global effects of such statements as `ODS`, `TITLE` and `GOPTIONS` should be unwinded at the end of the local scope to restore their status to the default states.
- IML programs should not accept the default variable names given by the READ statement from an external data set. Such practice is an invitation to trouble as name conflicts emerge.
- Use ODS in formatting results.
- Vectorization is important in IML. Aside from the reduction of interpretation costs, this practice will maximize the performance gain from BLAS if available. Therefore, some seemingly inefficient codes may actually be more favorable for such an interpretive matrix language.

[7]The AUTOCALL facility comes handy here so that there is no need to provide another mechanism to resolve module definitions.

- Use `filename()` function to have system generate file reference, which will avoid creating conflicting reference as `FILENAME` statement does. Release a reference immediately after finishing its use.
- Do not use numeric or string literals inside any function or method unless these values are a part of an algorithm or inputs/outputs for a SAS function.
- Use "`RUN;`" and "`QUIT;`" to conclude each `DATA` step and `PROC`, respectively.

3.3 Error Handling

- Programming exceptions or traps are no substitutes for the error checking at the user interface level.
- Normal operations should not cause any avoidable warning or error in the SAS log.

4 Style

Formatting style always reflects personal tastes more than professional requirements. But consistency is the dominant principle.

4.1 Comments

It is preferred to have a tool that can extract embedded comments from the source codes to generate user documentation. Such a tool makes it possible to write comments once to satisfy the needs of both end users and developers. No matter whether a tool is used or not, the comment style has to be consistent.

The comments regarding macro codes should be enclosed with "`%*;`". The comments for SAS statements are written in the fashion of "`*;`". Particularly, the general comments for a procedure or data step should immediately follow the openning statement of `DATA` or `PROC`. The general description for functions or methods should be enclosed with "`/**/`" and immediately precede the corresponding defintion.

4.2 File Prolog

If CVS or another version control system with compatible support for keyword substitution is used, the file prolog should look like

```
/*
 * $Id$
 */
```

and must be followed by the descriptive comments for the primary macro or class in that file. This will create all the necessary information for the revision history. Do not include Log in keyword substitution. Inside codes, string Id should be substituted for other representation to avoid confusing the version control system.

4.3 Formatting

Here are some rules in formatting programs.

- All SAS keywords and procedure options should be written in upper case.
- Use 2 spaces, not tabs, to indent codes.
- Macros and generated SAS codes should follow their own indentation levels.[8]
- Programs should not exceed 78 columns usually and never exceed 80 columns.
- A `newline` is required at the end of a file.
- Use blank lines to separate logical groups within functions and methods.
- Put one space in both sides of an operator and none immediately after "`(`" or before "`)`".

[8]Mixing generated codes with macros makes it difficult to read no matter how the indentation rule is set. This rule relies on the leading percentage sign for macro keywords to visually separate them from generated codes, while the indentation highlights the hierarchy nature of codes in their own styles.