Introduction

A. The SAS* Macro language

The SAS user community has had eight years of experience with the SAS Macro language. It was certainly a great step forward, but my developing understanding of the language has been leading to growing dissatisfaction. Like many others, I find many macros unreadable, even if I wrote them myself. I am also uneasy about the vulnerability of most macros to unexpected input.

I have come to conclude that much of the difficulty in writing and debugging macros is due to the syntax of the language. The Macro language provides some rarely used advanced features at the cost of increased complexity and decreased reliability in more common applications. In the words of one skilled macro programmer,

"...I still wonder if the supreme power of the SAS macro facility may lead other users into the same Catch-22, where disaster is just another ampersand away." (Kretzman 1986, page 515)

B. The SAS Screen Control Language

Screen Control Language (SCL) has been available for three years. It is marketed as part of SAS/AF* software and SAS/FSP* software, which provide full-screen interactive capabilities. SCL also provides code-generation capabilities similar to those of the Macro language, and data manipulation capabilities similar to those of the DATA step and PROCs.

The design of SCL responds to lessons learned from experience with the Macro and Data Step languages. This paper contends that these improvements often justify the use of "Screen" Control Language in the development of large complex systems, even those with no interactive component.

This paper also provides instructions for executing SCL in batch environments, which to my knowledge have not previously been described.

The role of control languages in the SAS System

A. Control languages

In modern programming languages (such as PASCAL or C), programs are built from subroutines which may themselves be built from other subroutines. All subroutines are written using the same structure.

In contrast, the fundamental building blocks of the SAS System are invocations of procedures (including "PROC DATASETP") which are not themselves built from similar procedures. A different language must be used to control execution of the procedures. Both SCL and the Macro language can fill this role.

I am using the term "control language" to refer to those components of the Macro language and SCL which provide overarching control of procedure steps, including:

- conditional execution of procedures,
- repeated execution of procedures,
- manipulation of global memory (macro variables),
- packaging of a series of procedure steps into a single unit, and
- dynamic generation of procedure code.

The particular importance of dynamic code generation within the SAS System stems from the run-time compilation of SAS procedure calls. In most programming languages, instructions and varying data are passed to subroutines via arguments which are evaluated during execution of the program. SAS uses much more complex syntax to provide specifications to its procedures, yielding increased power and clarity. In effect, the entire block of code for a procedure invocation provides its arguments. Dynamic code generation provides the means for the executing control language program to supply specifications for each procedure call.

B. Choosing between SCL and the Macro language

Although SCL is primarily marketed as a tool driven by user interaction, while the Macro language more typically obtains its parameters from other programs, the dividing line has increasingly begun to blur. Screen Control Language programs need not have associated screens or run in interactive environments, and the Macro language now provides a screen display facility of its own.

When selecting the software platform upon which to develop a system, two factors encourage a focus on one language, rather than a mix:

1) the difficulties and risks associated with interfacing different languages, and
2) the maintenance costs and potential inconsistencies from implementing the same shared modules in each language.

It is therefore a question of which language is best suited for a particular application. The obvious choice for full-screen interactive front-ends is SCL. Once this decision has been made, the factors above suggest using SCL for internal processing modules as well. Ideally, the same internal modules could be used in both interactive and batch environments.
C. Control languages for batch environments

In my opinion, seL is sufficiently superior over the Macro Language to warrant consideration of SCL as the primary control language for large complex systems even if they have no interactive component at all.

The main drawback is that the specifications of SAS/AF and SAS/FSP do not provide for batch use of SCL. I have been very interested in finding ways around this limitation, and have met with some success.

There is, of course, a catch. The key to success is a procedure which is undocumented and not intended for this purpose.

Batch use of Screen Control Language

A. Invoking SAS/AF

There are three methods of invoking SAS/AF programs, each restricted to a particular environment.

A.1 AF Display Manager command

To invoke an AF system from Display Manager, use the command:

AF G=libref.catalog.entry.type

on any command line.

Arguments cannot be passed to the program. Information needed by the program may be obtained from a screen associated with the program, or read from macro variables or other data objects.

The AF command may be submitted to the Display Manager channel from Program channel code using the DM statement:

DM 'AF C=libref/catalog.entry.type' AF;

[I use the word "channel" to describe a route through which code may be executed. The most common channels are the Display Manager, Program (i.e. DATA step and PROG), and SQL channels. Commands may be directed to any of these channels from SCL.]

Display Manager commands (including the AF command) cannot be executed in batch environments.

A.2 CALL DISPLAY and CALL GOTO

These two statements are used with SCL to transfer control to another SCL program. Parameters can be provided on the CALL DISPLAY statement to transfer data to the called program in accordance with the specifications of the ENTRY statement.

A.3 PROC DISPLAY

This undocumented procedure remains in version 6.06 to provide upward compatibility with version 5 SAS/AF. The primary documentation for version 5 PROC DISPLAY is of the SAS/AF User's Guide, Version 5 Edition (pages 69-74). The only reference I have found in version 6 documentation is a statement in the "Changes and Enhancements" section of the SAS/AF manual (page xxviii) that PROC DISPLAY is still honored with version 5 applications.

The syntax for using PROC DISPLAY in batch is simply:

PROC DISPLAY C=libref.catalog.entry.type; RUN;

Note that if you are running SAS version 6, you must use version 6 AF catalogs.

The documentation does not discuss use of the procedure in batch environments (nor, of course, the execution of version 6 SCL programs). PROC DISPLAY will run such programs in batch provided that they:

1) Have blank screens, and
2) Do not use SUBMIT CONTINUE.

These conditions are further discussed below.

A.4 Limitations on invocations

Using PROC DISPLAY, AF can only be invoked from a program at a step boundary. The interactive AF command (submitted through the DM statement) is more flexible and can be invoked at a statement boundary.

Neither PROC DISPLAY nor the Display Manager AF command can run more than one AF application at a time, or have more than one window simultaneously active in that application. In general, it is not possible to invoke AF from code submitted using SUBMIT CONTINUE. A limited workaround is available based on the fact that two simultaneous AF applications may be invoked by invoking one application with the AF command and the other with PROC DISPLAY.

In contrast to AF, macros may be invoked in the middle of a statement or from code that is being generated by another macro. The ease and flexibility of macro invocations is the primary reason that I still prefer macro programs for simpler "quick and dirty" tasks.

SCL systems need to be designed differently from macro systems, separating the AF programs from the submitted code. This task is simplified by the version 6 SUBMIT CONTINUE feature, which permits the AF environment to be maintained while the submitted code executes.

B. SCL programs with blank screens

SCL programs may be written without an associated screen simply by leaving the screen definition blank. All of the code may be placed in the INIT section of the code, because the other sections relate to the display of screens. Empty MAIN: RETURN; and TERM: RETURN; sections may be included to avoid compilation warnings.

In Display Manager, a default "status window" is displayed if no screen has been associated with the program. This is somewhat unattractive, but I know of no way to suppress it.
C. Code submission using PROC DISPLAY

Because PROC DISPLAY executes as a procedure running in the Program channel, no other code submitted to the Program channel can be run until the current procedure is completed. As a result, PROC DISPLAY cannot process SUMMIT CONTINUE statements correctly.

Unfortunately, AF does not automatically recognize when this problem occurs. Instead, the submitted code accumulates in the PREVIEW buffer, and is executed all at once upon exiting PROC DISPLAY. The exception is that if the final SUMMIT block specifies the CONTINUE option, it is executed in the wrong order, prior to the contents of the PREVIEW buffer.

This is obviously undesirable. To avoid this problem, use the SCL function SASTASK to determine whether SCL has been invoked by a SAS procedure. If so, use the SUMMIT statement rather than SUMMIT CONTINUE. Generated code will be held in the Preview buffer and automatically executed when the application terminates.

A PROC DISPLAY reference can be included in the generated code to restart AF after execution. This method is awkward, but was standard practice before version 6 introduced the SUMMIT CONTINUE statement.

For the sake of consistency, SUMMIT commands can also be used when SAS/AF is invoked through the AF command of Display Manager. In this case, the Preview buffer can be executed prior to termination of the application by using the SUMMIT CONTINUE statement.

The limitations on the SUMMIT command using PROC DISPLAY are not as consequential as they might be because
1) SUMMIT CONTINUE works correctly when submitting code to the SQL* channel, and
2) SCL has substantial data management capabilities of its own.

Because of these factors, it is often not necessary to submit code to the Program channel.

A comparative review of the SAS Macro Language and Screen Control Language

A. Advantages of SCL over the Macro Language

The most important advantage of SCL is that SCL avoids the unpredictability of SAS macros. Macros are difficult to write, debug, and maintain not only because of the complexities of the task they are undertaking, but also because the design of the language makes simple programs difficult.

SCL adheres to six principles that are frequently violated in the Macro Language:
1) Local actions have local consequences.
2) Programs treat all valid data values according to the same rules.
3) It is possible to review the data values entering a given module.
4) The algebraic properties of arithmetic are maintained.
5) Code generation can be avoided when it is more straightforward to manipulate data objects directly.
6) Code submission is under direct control of the programmer.

These six principles make SCL programs more predictable, more reliable, and easier to understand than SAS macros.

A.1 Local actions should have local consequences

This makes it easier for the programmer to understand how a given block of code will function in a program.

A.1.a Context-sensitive defaults

The term "user-friendly" was coming into vogue while the Macro language was under development. One definition of this term is the provision of extensive defaults appropriate to the current context. This introduces interrelationships between the code and its context which are not problematic in ad hoc use but which complicate maintenance of complex systems with long lifespans.

The obvious solution is to specify defaults explicitly, so they do not depend upon the context in which the code is used. In some cases, this leads to inefficiencies, such as using the %EVAL function when it is already implied. In more serious cases, explicit specification of defaults causes errors.

One of these cases is the specification of empty parentheses when invoking a macro with no arguments. If a macro has been defined with no arguments, then any parentheses following its invocation are treated as open code. If arguments are later added to the macro definition, then the same parentheses are treated as part of the invocation. The correct syntax is dependent on the macro definition, and if it changes, the calls may need to be changed. To avoid this problem, you can quote parentheses intended as open code.

In Screen Control Language, program calls always specify the parentheses.

A.1.b Same code acting differently in different contexts

Code debugged in one context may not work correctly when transported to another context. For example, comment statements containing unmatched quotes work in SCL or open code but not within macros. CARDS statements also do not work within macros, but can be handled correctly in SCL by using the CONTROL ASIS statement.

Macro expressions are evaluated differently depending on the macro statement in which they are used. If the expression must have a numeric expression, then a %EVAL function is implied. If a character result is valid, then the expression will be evaluated as character unless a %EVAL function is specified explicitly. This requires the programmer and reader to be attentive to the rules of each macro statement. In SCL variables are defined as character or numeric and act accordingly.
Languages such as Screen Control Language and the DATA Step language allow individual terms of an expression to be freely substituted without disturbing other parts of the expression. In the Macro language, changing part of a word may alter the number of scans made of the entire word, thereby changing the number of ampersands and dot delimiters that are required. For example:

```
MACRO A.J 'A' | 'J'
AA2..J A2 | 'J'
&A1.A biên J A(1) | 'J'
&AA&I...J A(J(K)) | 'J'
```

These problems are derived from the design decision to have the dot following a macro variable reference be optional, so it could not be used to indicate nesting.

A.1.c Same code acting differently in different referencing environments

It is often not feasible to completely retest each module in each location. Problems that vary according to the environment in which the module is called can be difficult to detect and debug.

In the Macro language, all variables from the calling environment are accessible from the called subroutine unless superseded by a symbolic parameter or local variable. Experienced programmers know to explicitly declare local macro variables as such, in case the macro is ever called from an environment in which that variable exists. In SCL, no one makes this mistake, because all variables except for parameters are automatically local.

One use of external variables is to pass the name of an external variable to a macro rather than the value of the variable. The called macro can then update the external variable, as in:

```
%macro INIT (NAME);
  %let NAME = 0;
%mend INIT;
```

This works fine as long as no one calls

```
%INIT(NAME);
```

in which case access to the external variable is blocked by the name of the local parameter. There is no warning, but the external variable remains unchanged. These limitations increase if the macro uses additional local variables.

A commonly used technique for creating a new macro variable employs the IGLOBAL statement:

```
%macro NEW (NAME);
  %global &NAME;
  %let &NAME = 0;
%mend NEW;
```

The problem is that this new variable is not defined at the level of the calling program, but at the outermost level. If a global variable of this name already exists, its value will be destroyed.

In Screen Control Language, program calls are transactions directly affecting only the parameters of the call, minimizing the risk of unintended side-effects.

A.2 Programs should treat all valid data values according to the same rules.

This assures the programmer that the program will operate correctly over the entire range of valid input.

In the Macro language, the programmer must anticipate every type of data that might cause trouble in that particular context. It is my impression that very few programmers consistently write bombproof macros that can deal with unexpected input.

In Screen Control Language (as in the DATA step) data values are isolated from program code. While some of the tricks of macro language are impossible, the quoting functions of SCL are trivial, and the programs consistently work.

A.2.a Numeric comparisons

The same macro expression will compare some values using numeric rules, and other values using character rules. Integer numbers are compared numerically. For example, 002=2. Decimal numbers are compared as characters. This will give the expected result only if both values have decimals and the number of leading zeros and trailing zeros are the same:

```
1.23 = 1.23, but 01.23<1.23 and 1.230>1.23.
```

SCL uses numeric comparisons for all numeric variables.

A.2.b Indirect macro variable references

The Macro language allows data values to contain references to macro variables, which are then recursively resolved. This unusual capability can cause chaos because it introduces additional scans of the word. If an expression sometimes requires more scans than other times, it is impossible to use a consistently correct number of ampersands and dot delimiters in the program. The only solution is to resolve the variable value before using it in the complex expression. It is therefore unsafe to use this capability unless the procedure has been designed with this intention.

A.2.c Special symbols and reserved words

Macro calls, macro variable references, expressions, and label identifiers are all evaluated at execution time. This permits many tricks such as dynamic tricks of macro language are impossible, the quoting functions of SCL are trivial, and the programs consistently work.

A.3 It should be possible to review the data values entering a given module.

When debugging is necessary, this aids in determining where the problem occurred.
A.3. a Quoted values

In order to deal with various problems, the Macro language provides quoting functions that work by recoding special characters to other unused codes. These recoded values can be safely manipulated without needing to use the quoting function again.

Both quoted and unquoted values print and compare identically, so you can't tell what is quoted when you are debugging.

In SCL, the characters printed are those used internally.

A.3.b Macro variable table contents

SCL provides a Debugger facility that allows executing a program one line at a time, pausing at a particular point in a program or when a specified condition is satisfied.

The Macro language has nothing like this. The most crucial gap is the absence of a command such as EXAMINE ALL, which dumps the value of all SCL variables in the current environment. This has been on the SASware Ballot* for years.

A.4 The algebraic properties of arithmetic should be maintained

This provides confidence that the implementation of an algorithm will not affect the outcome.

The Macro language only handles integer numbers. If a division operation produces a noninteger result, it is truncated to an integer:

%put %eval(7/3); prints 2.
Because intermediate results are also truncated, fundamental algebraic laws such as commutativity may not hold even if the end result is an integer:

%put %eval ( (7*6)/3); prints 14.
%put %eval ( (7/3)*6); prints 12.
This is more difficult to debug if the arguments are macro variable references and the operations are divided into several steps.

SCL provides a real numeric variable type. Unlike the Macro language, missing values are supported.

A.5. Code generation should be avoided when it is more straightforward to manipulate data objects directly.

This reduces program complexity and is more computationally efficient. It also permits the program to determine the exact nature of errors and respond appropriately.

The Macro language cannot directly manipulate data objects other than macro variables. The appropriate Program channel code must be generated. Information is then transferred back to the macro level via CALL SYMPUT. The same technique is required to access functions other than the five (excluding quoting functions) that are supported at the macro level.

SCL can directly manipulate an even wider range of data objects than the Program channel, including SAS system options, host system options, titles, footnotes, external directories, SAS catalog directories, SAS catalog entries, and SAS datasets and indexes. Over 150 functions are available.

In many respects, SCL has capabilities superior to those of the program channel, including

- maintenance of open files (and associated locks) across steps,
- programmable response to many errors,
- execution-time specification of data objects (allowing more precompilation)
- linking of independently compiled programs at execution time,
- dynamic execution-time specification of WHERE clauses,
- programmable handling of input and output buffers.

A.6 Code submission should be under direct control of the programmer

In the Macro language, generated code is executed as soon as a complete unit is specified. This can cause confusion, because macro code and open code are interwoven and executed alternately.

SCL generates code into a PREVIEW buffer which can be edited, saved, or canceled prior to submission. The code is executed when the programmer specifies.

B. Advantages of the Macro Language over SCL

A primary advantage of the Macro language for batch use is that unlike SCL, it is designed and documented for these applications.

B.1 Named parameter syntax

Macros may be called using named parameters, which are easier to read, may be specified in any order, and will take the specified default value if omitted.

SCL only supports positional parameter syntax. Modifications of existing programs may require changing calls of those programs in existing code. The lack of support for named parameters is a serious limitation, but a useful work-around is to write macros that accept named-syntax calls and generate the corresponding SCL call. Macros may be used to generate SCL code at SCL compilation time.

B.2. Subroutine libraries

The SASAUTOS macro library facility allows searching a list of libraries in priority order. This makes it possible to have a set of standard macros which can be superseded by custom macros for specific projects: a limited form of the "inheritance" concept of object-oriented programming.

SCL does not provide such a capability.

B.3 Flexibility of macro invocations

This was discussed earlier.
B.4 Ease of program editing

SCL programs can only be edited using the full-screen interactive procedure PROC BUILD, even if there is no interactive component to the system. Macro language programs can be edited using any text editor.

B.5 Automatic error handling

If a DATA step or PROC generated by a macro encounters an error, it is automatically handled by this system. While the method in which it is handled may not be the most desirable, it requires no attention on the part of the programmer.

Most error handling in SCL is the programmer's responsibility. This level of detail contributes to the typically messy appearance of SCL programs. A program may continue processing after an error has been encountered, unless instructed otherwise.

B.6 Lower-level programming

DATA steps and PROCs act at an abstract logical level. For example, the programmer need not be concerned with opening an input file, creating an output file, fetching the next case, transferring information to the program data vector, updating the output file, detecting the end of the input file, and closing the files.

In SCL, these operations are the programmer's responsibility and under his or her control. It is possible to accomplish tasks that cannot otherwise be done in the SAS System. On the other hand, if you want to do something routine, this level of detail is a nuisance. Of course, you could develop a set of standard SCL programs or SAS macros to generate commonly used code.

C. Summary

If you have large amounts of generated code with limited control code, macro is a good choice. It is also appropriate when building a set of tools for ad hoc use in developing other macros or Program channel code.

SCL is a good choice for the development of large complex systems that will require substantial maintenance. SCL programs intended for use in batch environments should avoid submitting code to the Program channel. SUBMIT SQL and the SCL data management facilities should be used instead.

Wish List for the future

The following enhancements to SAS would be extremely helpful for developing systems using Screen Control Language in batch environments.
- Allow all methods of invoking SAS/AF to accept parameters. Support named-parameter syntax throughout, including specification of default values.
- Provide a CATSEARCH statement analogous to the version 5 LIBSEARCH statement, but using a specified list of catalogs.
- Support the SUBMIT CONTINUE statement in batch environments.
- Support ON units, which would define code to be executed whenever the indicated SAS return code occurred. For example, ON ZSYSRC(SENOLCK) DO; RETURN; END; This would simplify SCL error handling.

These enhancements would make 'Screen' Control Language the preferred choice for systems control and code generation throughout all modes of SAS operation.

Acknowledgements

I would like to thank Grant Blank for his comments regarding an earlier draft of this paper.

References


SAS. SAS/AF, SAS/PSF, and SASware Ballot are registered trademarks of SAS Institute, Inc. SQL is a trademark of International Business Machines Corporation.

The author may be contacted at:

Andrew A. Norton
Trilogy Consulting Corporation
850 S. Greenbay Road
Waukegan, IL 60085-7076
(708) 244-9520