Abstract
In 1982 the SAS Institute introduced to SAS programmers the MACRO facility and language. Approximately 6 months after this introduction a strange and until now, unidentified disease began to pervade the SAS programming community. Left untreated MARCOITIS causes indistinguishable black boxes disguised as MACROS in SAS programs throughout the nation.

Macros can make life as a SAS programmer easy or can become a nightmare of maintenance and support. There have been many GOOD MACROS and MACRO applications that have been written. Unfortunately, the BAD and the OBNOXIOUS far outweigh the GOOD.

It will be intent of this paper to arm our community with the tools to identify the symptoms of MACROITIS, create preventative treatment, and for severe cases introduce a new innovative vaccine called MACROSENSE.

Identification of Macroitis
In order to correct or prevent the disease we must first learn how to identify some of the signs or symptoms of MACROITIS. Our clinical trials have involved subjects over a ten year period and have been able to identify the behavior exhibited by programmers infected with Macroitis.

Remember, even though you may be able to identify macroitis, the programmer who is infected must first admit that she/he has some of the symptoms.

Macroitis Symptoms
1.) Grinning at the monitor.
2.) Every 3rd or 4th sentence is "I've got a macro that will do that!"
3.) Substringing a macro at least 3 times to obtain another macro.
4.) Creating a macro to check other macros.
5.) Can't macro SQL --- yet.
6.) Neighbors report that these individuals are loners and quiet types.
7.) Strange ritual beliefs that macros get smarter.
8.) Cannot write a sentence or a report without a % or &.

The Good MACROS
What constitutes a GOOD MACRO
There really are some good even great macros out there in SASland. But how do you identify what is a good macro? Good macros tend to make MACROSENSE. One simple yet good example is a macro called AUTODOC (Figure 1).
There are several good points that need to be identified in this macro. The first and best point is that there is DOCUMENTATION. By looking at this macro program the programmer can identify who wrote it and when, what it does, what the usage of the parameters is, and what the default parameters are. It is clearly usable both in a data _null_ or any proc that may be used. Even the name AUTODOC suggests the usage of the macro.

**Report generation**

MACRO's can save immense programming time and typing effort by simplifying tasks. For example, if you had to produce a report for 50 different study protocols each displaying race and sex, it would make sense to create a macro to do that. Certainly a programmer could copy the program 50 different times, but that would be inefficient. Besides, this macro IF thought out well could be used for other variables, as in Figure 2.

Calculations

Other macros that makes sense are conversions for units of measure. Temperature conversions from celsius to fahrenheit or fahrenheit to celsius would be a SENSIBLE macro to have around. Other examples would be Weight conversions from pounds to kilograms or height measurements from inches to centimeters. A macro for both of these makes SENSE because it insures that these conversions are being done EXACTLY the same way every time and that there is no room for employee or consultant creativity.

**Output Data Sets**

Some companies depend on an output data step that contains only the data displayed on a particular report. The user of the data also tends to want the data in the same order as the report so that he/she can "print" the data and compare it to the report (as though it would magically change), or manipulate the data in another way to obtain a different comparison.

This is another situation where a macro makes SENSE and is useful. Also note in Figure 4 the use of documentation which fully explains the use of this macro.

The Bad MACROS

Passing unknown parameters

Nothing is worse in programming then coming upon a
MACRO that works in a black box. A hole in the program were the "magic" takes place. For example, the program that you are working on calculates the age of the subject as in Figure 5.

```
DATA DEMOG;
  SET IN.DEMOG;
  BIRTH=DOB;
RUN;
%CHECKAGE(,);
```

Figure 5

What are the parameters being passed? Will the macro just check the age of the subject or will it actually calculate the subject's age? This is a BAD macro because there is no internal program documentation describing the process. A simpler way to use this macro would be to use one or two lines of program documentation like this;

* CHECKAGE uses the _last_ data set created and birth to calculate the age of the subject;

These two lines of comments tell the programmer that the macro call must follow the previous data step and what action is being taken by the macro. There is no longer a "black box" and the "magic" no longer exists. BAD macros need not continue to thrive. Simple steps like internal documentation can be taken to turn them into GOOD macros.

Passing over 50 parameters

The day has come to set a limit for the amount of parameters passed to one macro. Let's be honest, why do some macros pass more parameters than code? Does the macro start out that way or is it the result of a mutant strain of some %do %until gone wild? Actually the answer is neither.

Most macros start off relatively simple with maybe 5 or 6 parameters. However, the user says "Gee that's great! But could I also do this?" The macro is modified. This is called the user-loop. Our clinical studies have determined that at least 7 parameters are added to the typical macro every time it processes the user-loop. The user-loop never ends. But this loop does not have to perpetuate BAD macros, making them worse and finally passing 200 to 300 parameters to what started out to be a simple proc print!

When the original purpose of a macro becomes obscured, it is time to RETHINK what the user really wants and how to obtain that result.

Macros that produce a major report or become part of a system must be DESIGNED and not built by an iterative user-loop. Design will begin the process of controlling macros throughout your organization.

The Obnoxious MACROS

These macros are the plague of SAS programmers. They can, however, be classified and therefore can be dealt with when encountered. These classifications are as follows:

1.) Unreadable macros.
2.) Macros that hide the process taking place in the program.
3.) Macros that obscure the process taking place in the program.
4.) Macros that create logicals(true -False) and do not use them.
5.) Macros that "save" CPU time.
6.) Macros that do not use "normal" naming conventions.
7.) Macros that call macros that call macros that call macros.
8.) Macros that do not do anything, but are placed in a program where a critical calculation is taking place.

Unfortunately we will not be able to deal with each one of these classifications. But they are listed here as a warning of a potential danger in any program. Dealing with the obnoxious macro and macros in general will be in the Preventative section. The following situation incorporates at least 4 out of the 7 classifications listed above.

You are the programming manager with a leading pharmaceutical that is about to file an NDA within 10 days. The principal statistician has found a major error in your demographics table. (Males and Females are overlaying each other without the use of safe spacing). Since most of the consultants hired to write the programs are long gone, you decide to make the modifications yourself. The program is located as DEMOG.SAS, you edit it and this is what you see:
After the shock has worn off several questions come to mind. Where does the real program live? What variables do the parameters represent? What are the default parameters? WHO WROTE THIS (and where can I find him/her)? Is there any documentation anywhere?

After searching through 5 macro libraries, the actual DEMO program was found. However at the end of the demog program where the data _null_ was to be found, another macro "%null" was there. This macro led to another program where the data _null_ lived.

After looking at this program, you found out that this %null was responsible for the print of approximately 18 additional reports. If you made a change here you would have to rerun one-third of your NDA. Knowing that there would not be enough time, you re-write a DEMO program barren of macros.

Another prime example of an OBNOXIOUS macro was one that I found in the middle of a lab program. This program was calculating the time between the dose of a drug and the onset of a particular adverse event. The process was critical and was developed as an answer to a FDA question. In the middle of this process was the following macro:

```
%TIMECALC;
```

Thinking that this was a key element in the program, I researched it only to find that this macro resolved to:

```
OPTIONS FULLSTIMER;
```

It did nothing. It did obscure the process. It was obnoxious.

These are prime examples of MACROS that have had no control, no standards, and certainly show a lack of monitoring. These OBNOXIOUS macros should have never been created. The only way to circumvent this occurring again is by some PREVENTATIVE TREATMENT!

Preventative Treatment

How to Control MACROS

In order to control the MACROS in your organization development of Macro Standards is important. These standards should be part of SAS programming standards already in place. Some of the tried and true standards follows.

Central library

Create a ONE central repository for all macros. The library then can be accessed by all. NO MACRO should be in an individual’s program or library.

Naming conventions

Use names that help describe the process. Some of the examples in this paper are AUTODOC (auto-documentation), DEMOGRPH (demographics), or FREQ2VAR(frequency-2 variables).

Data set names used within the macro should be consistent. For example if the data set contains drug information RX001 isn’t as descriptive as drugdata.

DOCUMENTATION

All macros should have External User Documentation. Internal Documentation (within the macro code) should indicate what the macro does and why. It should contain the value of the Substitution Parameters, Date written, Author, and what the Default parameters are.

Programs using the macro, should briefly detail what the macro is doing.

The documentation should also indicate whether the macro executes, within a data step or procedure or independent of a data step or procedure.

Lastly, Consultants or employees who refuse to conform to these standards should be treated with extreme suspicion. Are they doing ANYTHING you want, or just carving their own niche within the organization?

Conclusion

Using macros can save time and be an asset to any programming department. It is how they are created, what they do, and where they are stored that needs to be controlled. Part of the control is simple COMMON SENSE or in this case a VACCINE called MACRO-SENSE

MACRO-SENSE VACCINE:

1.) It must make SENSE to macro it.

2.) The macro should bring consistent results and save time.
3.) The MACRO should actually do something.

4.) The MACRO must be maintainable by more than 1 person.

5.) The MACRO should provide an explanation of all parameters.

6.) The MACRO should provide a clear and concise summary of what it does.

7.) The MACRO should be able to be used several times.

8.) The MACRO must adhere to programming standards.

9.) The MACRO must be planned with an obtainable result.

10.) The maximum amount of parameters must be set.

11.) All macros should have a 5 day waiting period. If the programmer can still understand why he/she wrote it, then it can be used.

Acknowledgements:

SAS® is a registered trademark of SAS Institute, Inc in the USA and other countries. ©indicate USA registration.

I would like to thank Faith Renee Sloan and Adrienne Koranda for the use of their "GOOD" macro code, and Ms. Joanne Laffer and Mr. Thomas Camp for their technical and editorial review of this paper.