PharmaSUG 2018 - Paper AD-01

Developing Large SAS® Macro Applications

John Ingersoll, Rho, Inc.

ABSTRACT

The SAS macro language is used every day by statistical programmers for repetitive code and substitutions in programming data sets and displays. But it can also be used to develop larger applications more similar to a traditional software package than what many think of when they hear "SAS macro." However, most companies don't have a "SAS Macro Application Development Department" and many talented and experienced SAS macro programmers don't have experience in a traditional software development environment. This paper presents the development of a validated application written solely in the SAS macro language, containing 124 macros. It discusses the challenges, considerations, and solutions to issues experienced in the development process, with attention to the fuzzy distinction between a SAS macro and a software application or system.

INTRODUCTION

The focus of this paper is on higher level lessons learned in the development of a large macro application, with some specific principles and examples to follow. The application, "RhoTables", is not the subject of this paper, since it is a proprietary system developed for in-house use. However, it is used as the example and presented to the extent necessary to convey a range of issues including validation, versioning, system/application design, and staffing.

THE APPLICATION

RhoTables™ is a reporting application that is an alternative to PROC REPORT and ODS or other methods of generating tables and listings as RTF. It was initially developed with three major goals: to simplify the creation of statistical displays as RTF; provide a valid and reliable means to programmatically validate displays through independent double programming without manually comparing output; and to overcome certain limitations of PROC REPORT. It has grown considerably over the years, from a handful of user written macros to a fully self-contained system. In its current (4th) manifestation, it contains:

- 124 SAS macros
- 193 possible arguments to three main macros
- Unlimited column layout definitions with 5 parameters each and 25 additional options
- 25,060 total lines of code (13,042 executable, 7,816 comments, 4,202 white space)

Despite its complexity as a SAS macro system, it is relatively straightforward to use, with macro calls typically containing only a couple of arguments, and the ability to define parameters on a study by study basis.

HISTORY:

2003: Version 1 was an unnamed tool of about a dozen macros that generated RTFs in a single defined format and allowed for programmatic independent double validation, our initial goals. These initial macros were essentially a proof of concept initiative and a beta test for the methodologies employed, used on a limited number of studies.

2004: Version 2 grew to 33 macros and began to look more like an integrated system. However, the macros remained as individual macros in an autocall macro directory, along with other macros that were available for all SAS programmers to use.

As it became recognized as more than "just some macros" it acquired the name "RhoTables", in the spirit of prefacing our application names with "Rho." In order to delineate between improvements that would significantly enhance productivity and things that would be nice for programmers, we adopted the 80/20 rule, focusing on getting the "biggest bang for the buck." Repeating and reminding people of this principle was an almost daily occurrence, necessary to keep things focused, prioritized, and progressing.

Major additions included to the ability to generate RTFs with or without the use of MS Word tables, and to produce text output in a traditional SAS manner (i.e. *.lst files).

2006: Version 3 was a comprehensive rewrite as a cohesive system, rather than a collection of macros that work together. The 56 macros were designated "RhoTables3," to distinguish from version 2. In an effort to create a self-contained system, the macros were packaged in a compiled macro catalog, with only an initialization macro remaining in the general autocall directory.

In addition to adding functionality, significant effort was put into testing all parameters for valid values and consistencies, trapping error conditions, and terminating cleanly with meaningful messages to the SAS log. The ability to define default parameters on a study by study basis was added, as well as integration with other tools, specifically our Project Tracker database which stores display titles and footnotes.

From the beginning, we have strictly held to the rule that RhoTables must be 100% backwards compatible within a given version: any program written before an enhancement or update will run without modification and produce identical results after it (and pretty much forever). Version 3 was updated and enhanced over the years, filling in some of the 20% functionality remaining from the previous 80/20 rule as resources allowed. However, we found that it was not realistic to incorporate some desirable functionalities without breaking the backwards compatibility rule, so these items sat in a wish list.

2014: Version 4 was another significant rewrite with changes to the underlying code structure and metadata elements, necessary to accommodate the desired features and functionality that were incompatible with version 3. Therefore, RhoTables4 was largely undertaken in response to the backwards compatibility rule while providing additional functionality that was waiting in the wish list.

One thing we discovered with version 3 is that using a compiled macro catalog provided few benefits and created a lot of headaches. On occasion we are required to deliver executable SAS programs to clients, and the SAS catalogs were often incompatible. While SAS datasets are wonderfully portable, catalogs, unfortunately, are not. Internally, we came to have a mix of 32 and 64 bit machines, requiring us to maintain two versions of the same catalog. Our solution for version 4 packages all the macro definitions into a single program file that is %included as part of the initialization routine.

FUTURE:

A version 5 is highly unlikely. Version 4 is about as mature as a SAS macro system can practically get, and the cost/benefit of adding additional functionality without backwards compatibility is dubious. We will continue to add functionality and features to version 4 where there is a positive cost/benefit assessment and backwards compatibility can be maintained.

DEFINE AND UNDERSTAND THE USER BASE

It is essential to define and understand the user base when undertaking a large macro system. Initially, 14 years ago, the application was developed as a tool for a self-contained department of about 15 SAS programmers. As such, it was assumed that users were pretty good SAS programmers, at least somewhat experienced with the macro language, and could either decipher the obscure SAS messages, warnings, and errors that would occasionally arise, or had a handy resource in a nearby cubical that could help out. Initially, this assumption worked, but over time it became problematic.

Today the potential user base is over 100 statistical programmers, statisticians, and research assistants spread across the company and with widely varying knowledge and experience in SAS and macros. In response to this change, more than half the code in RhoTables today is checking for valid and consistent specifications, an exponential growth. In addition, there has been a significant expansion of the user documentation and the development of training materials and sessions. And finally, functionally has been incorporated into the system over time that from the developer's perspective could be otherwise accomplished in SAS outside of the application. Therefore, it is important to anticipate future use and changes, in addition to the immediate need.

A significant take-a-way is that the simpler and easier the application is for the user, the more complicated it is for the developers. The 80/20 rule works for efficient development, but isn't necessarily well received by the users. Users will ask, "Why can't it do this or that?" and "It would be so much easier if …." Developers will respond, or at least think, "But all you have to do is …." It is important to anticipate this tension when deciding to leave out some features and functionality, and to allow the possibility of adding them at a later date.

VALIDATION AND PROCESS DOCUMENTATION

As soon as a SAS macro, or a group of macros, becomes identified as a package, application, system, or software with a name, it becomes subject to questions and scrutiny by others using criteria not typically applied to SAS programming. Over time, the identity of RhoTables has changed from "just a bunch of macros", to a "statistical programming tool", to a "system", all the while performing the same basic task—just with (significantly) more functionality, polish, and flexibility.

Validation has remained under the purview of our Statistical Programming SOPs, to maintain the flexibility accustomed to by SAS programmers and the requirements of daily production work. But with each iteration we have gone further beyond and borrowed more from generic software development practices and procedures.

The version 1 macros were validated under our Statistical Programming SOPs as individual macros used in display generation. They were not validated as a system, but as any other macro written for display production.

With version 2 we recognized that our Statistical Programming SOPs for macro validation were not very applicable to a complete system of macros, but on the other hand it was not a traditional piece of software that fell under a standard system development life cycle (SDLC) process. We handled this dilemma with a hybrid approach: we updated our SOP for validating SAS macros to include a more stringent process for systems of interdependent macros and validated them under those criteria. This included writing and executing test cases in a manner similar to a SDLC. However, we eased the requirements, relative to SDLC practices, for releasing updates and revisions. A stringent multi-step process of testing and re-validation would be impractical in our SAS programming environment, and we did not have those resources.

Version 3 underwent a complete validation similar to version 2 in process and scope. By this point RhoTables was recognized as an application or system beyond our Statistical Programming group and came under more scrutiny from auditors. Our QA department did an internal system audit which led to enhanced process documentation and record keeping, generally being well received by outside auditors. Typically, the most difficult part of an audit was hauling the banker's boxes full of test cases to the conference room.

The development and validation of version 4 reached further beyond the Statistical Programming group, including an IT project manager and contributions from our software development validation team. In response to previous audits and changing internal and external standards, we refined our approach to validation, moving closer to the SDLC model. Our software validation specialists wrote the formal validation plan, SOP exceptions and justifications, and provided guidance in documenting functional specifications and test cases. They also maintain the documentation.

This approach has worked well on the whole, but there has been a learning curve. Here are some pertinent lessons:

- Determine 21 CFR Part 11 applicability. Document your rationale and have it readily available to anyone who may be questioned about its applicability. While outside the scope of this paper, this is absolutely essential as it dictates all processes and validation that follow.
- If not familiar, read up on accepted System Development Life Cycle (SDLC) practices. You need to be familiar with the language, the processes, and be able to justify why you did or did not follow

standard practices when communicating with auditors, internal QA, software developers, and validation specialists.

- Write and maintain a Validation Plan, following an SDLC template. If your company has a standard template, use it. It is likely that there may be non-applicable sections: leave them in your plan with a short explanation of why it is not applicable. Simply removing them may have the appearance that you did not do something that you should have.
- Write functional specifications and test cases. For some applications it may be rather redundant to do both, and the test cases can suffice as your functional specifications. For versions 2 and 3 we did not separate functional specifications and tests cases, although we did in version 4. The most important thing is that you document your method and rationale.
- If the application is subject to being audited (yes it is), remember that it is essential to (a) document your processes, (b) follow those processes, and (c) document any exceptions. There can be flexibility in what you decide to do, and how you go about it, but you must document your process (say what you are going to do), and follow your processes (then do it). The most likely audit findings result because you do not have proper documentation or did not follow your procedures; findings are much less likely to be about what those processes are.

Another important consideration is the data used for validation. In a macro system like this, it is likely that many bugs will be related to data issues and values, so it is important to test, if not formally validate, with realistic data. Unfortunately, many applications (SAS or otherwise), appear to be tested with many implied data assumptions that don't hold up in the real world.

Actual clinical trial data may not be used to validate the application for both ethical and legal issues: patient confidentiality, contractual, the sensitivity of the data, and the sensitivity of the results. Our solution was:

- Testing during code development used a mock clinical database that reflects real clinical data. This is an internal database that we constructed and maintain for general testing of clinical applications without confidentiality concerns.
- Test case execution used constructed, non-clinical, data sets. We made the assumption that test cases would be "public" in the sense that they could be reviewed by people who were not authorized to view a specific clinical study, or studies.

VERSIONING AND UPDATES

One major concern is that RhoTables is used daily for time sensitive deliveries that our business is dependent on. And given the flexibility and seemingly infinite possible combinations of parameters and specifications, it is always possible that a user will discover an interaction that has not been previously tested or used, typically on the day of a delivery. In this situation it is not acceptable to delay a delivery while waiting for the application to go through a formal SDLC process of specification, revision, testing, validation, and implementation. Simply put, it needs to be fixed now--period.

Our solution is to specify the criteria and validation/testing requirements for three levels of updates to the application in the validation plan:

1. Major version: Any change that does not maintain backwards compatibility with the previous production version constitutes a new major version, unless it corrects the functioning of the system.

   Major versions require a new validation plan, functional requirements, and documented test case execution.

2. Revision: Significant changes to the source code, or changes that interact significantly with pre-existing functionalities of the system. Revisions may add new features or correct problems in the functioning of the system.

   Revisions require validation of new or modified macros, similar to individual macros under the Statistical Programming SOPs.

3. Maintenance release: Corrects a problem in the functioning of the system, or adds functionality that does not constitute a revision or major version.

   Maintenance releases do not require formal validation, but are tested by one or more of the following methods:

   Running malfunctioning programs to ensure issues are corrected.

   Running previously functioning programs to ensure backwards compatibility.

   Rerunning test cases from the validation of the major version to ensure backwards compatibility and to test for interaction with existing functionality.

   Creating new programs to test new functionality.

Version numbers are in the form of v.rr.mmm where "v" is the major version, rr is the revision, and mmm is the maintenance release, e.g. 4.03.011.

INTERNAL DOCUMENTATION OF UPDATES

Documenting updates and changes to the macro code is essential to the maintenance of the system. Since there are a multitude of possible combinations and interactions of parameters it is always possible to introduce a bug while fixing another. And because it may involve an unlikely or uncommon combination of specifications, it is essential to be able to track and identify when and where the relevant code was modified, sometimes months and several releases later.

We consider it excessively cumbersome, impractical, and unnecessary to maintain separate external documentation of all updates, but instead document any code changes internally in the macro program files, strictly adhering to the following rules:

1. All updates are documented in the header of the main calling program, in this case RhoTables4.sas. This is the master list of any and all updates to the system. A line is added to the program header under "Program History" that includes:

   - The date the revision or maintenance release is put into production. This must match the date stamp of any modified program files.
   - The name of the programmer making the change.
   - The new version number enclosed in square brackets, e.g. [4.01.000].
   - A brief description of the change.
   - The names of all macros that were modified to implement the change, enclosed in parentheses.

   For clarity, functionally different modifications that are implemented concurrently may have separate program history lines.

   Examples:

   ```
   23Feb2016 John Ingersoll [4.03.003] add RhoBlue and RhoGreen colors
   (rt4Formats, rt4RenderRtfNew)

   10May2017 John Ingersoll [4.04.027] fix for no title in an Intext table
   (rt4RenderRtfSec)
   ```

2. In all modified macro program files:

   - A copy of the program history line from RhoTables4.sas is placed in the header of each macro file, except that the names of all modified macros are dropped. This line is added only to macro program files that were modified to implement the change.
   - The version of a modified macro is the same as new version number of RhoTables4.sas, even if it is not sequential within the individual macro. Individual macros do not have their own separate version numbers.

- All modified or added source code is commented, to include the new version number enclosed in square brackets, e.g. [4.01.000].
    - Self-evident modifications may be commented with the version number only.
    - Individual lines may be commented with a trailing comment statement.
    - Short sections of code may be commented with a leading comment only.
    - Long sections of modified or added code are delimited with leading and trailing comments to indicate the extent of the modifications to the code.
- Deleted sections of code are replaced with a comment. Leaving commented out code in the program file is discouraged, as it usually more confusing than helpful.

3. The version numbers of all other non-modified macros will not change. Therefore, the last version number in each macro program file reflects the version of RhoTables at the time the last modification was made to that macro.

VERSION CONTROL

The steps for version control are consistent with our SAS macro SOPs, but go beyond those specified in recognition that RhoTables is a system of macros and not a singular one. Since we do not use version control software for SAS programming, we developed the following version control steps for developing RhoTables.

All development work and testing takes place in a separate development area on a different network drive than the production location. The production location has strictly controlled write access to prevent modification there.

A procedure was written (in SAS) to automate the installation of the application and updates:

- It reads the last version number from the header of RhoTables4.sas, the main calling macro.
- It reads all macro source files, removes comments and blank lines, and concatenates all macros into a single source file.
- It creates a new backup sub-directory named with the version and date in the form of `..\Version_4.rr.mmm_ddmmmyyyy`.
- The new version is copied from the development area to the newly created backup subdirectory, e.g. `..\Version_4.04.030_16OCT2017`. Both the individual program files and the concatenated system file are backed up.
- The new version is copied to the production location, overwriting the previous version. Only the concatenated system file resides in the production location.
- The install program verifies that the appropriate files were copied with the appropriate date/time stamps.

This approach ensures that a backup copy of the current version exists at all times and any inadvertent change to or deletion of production files may be immediately reversed. It also allows the ability to rollback to any previous version of RhoTables at any time. To rollback to a previous version, the appropriate backup directory is simply copied to the production location, and the backup directory of the version that failed is deleted.

This method also allows individual SAS programs to point to a backup directory and run any previous version of RhoTables. However, this should only be necessary, and temporary, if the backward compatibility rule is inadvertently violated.

SYSTEM REQUIREMENTS AND CONSTRAINTS

It is essential to define the platform(s), system requirements, and any dependencies upon other software. The following broad requirements and constraints were defined for version 4, focusing on portability and minimizing dependencies:

- RhoTables4 is entirely comprised of SAS macros.
- RhoTables4 requires only SAS to be functional. It is not dependent on any other software or system.
- RhoTables requires only base SAS products.
- RhoTables4 is operating system independent. No system dependent calls or piping of system commands are used.
- RhoTables4 will run on SAS 9.2 and all higher versions. No functions, statements, or features of SAS that are not a part of 9.2 are used.

  It may be necessary or desirable to modify this requirement at some time in the future. However, we must remain compatibility with any potential version of SAS that a client may be using at a given time.
- RhoTables4 utilizes Rho's Project Tracker database, retrieving and using Project Tracker data as specified by the user. It is not dependent on the Project Tracker application, but only utilizes a connection to the Oracle database. Further, it is fully functional without the Project Tracker, or with a Project Tracker database that has been exported to SAS data sets.
- RhoTables4 utilizes 3rd party software to generate PDF files from RTF documents. This dependency only affects the ability to create PDFs and is independent of all other functioning of RhoTables. It is an independent process with an integrated call from RhoTables for the convenience.

MINIMIZE ASSUMPTIONS AND DEPENDENCIES

We experienced several issues with version 3 that prompted some of the above requirements.

- It used an MS Word macro in the programmatic display validation module. This created problems with almost every update to MS Word, requiring emergency updates to RhoTables, despite the fact that the Word macro was incredibly simple.
- On occasion we are required to deliver executable programs to clients. There were often platform, version, and configuration differences that initially took some effort to work out.
- Because version 3 was implemented as a compiled macro catalog, and SAS catalogs are system and version dependent, we had to deliver the source code files and have clients compile the macros on their systems, creating additional work for both us and them, and negating any benefit of using compiled macros.
- Similarly, early versions contained a few operating system calls that occasionally created issues, even within our company. These were replaced with more generic SAS functions or techniques that are not platform dependent.

In cases where OS or version dependent code is unavoidable, enclose it in conditional logic, even if initially developed for a single defined environment. Computing environments change over time and successful applications have a tendency to grow beyond their anticipated use. This can also make it easier to identify dependent code through global searches.

For example, there have been multiple suggestions and requests to more fully integrate RhoTables with our Project Tracker over the years. Among other things, the Project Tracker maintains a master list of displays and stores titles and footnotes, which RhoTables reads and renders in the displays. However, this functionality is not a dependency, but rather a programming convenience for reliability and efficiency. To date, efforts by others to add features that would introduce a dependency of RhoTables on the Project Tracker have been successfully resisted in order to maintain its independence.

SYSTEM STRUCTURE AND PROGRAMMING

In the development area the macro code is written in individual files with 1 macro per program file, as is typical for any SAS autocall macro. However, in the production area the macros are concatenated into a single file, allowing the system to be fully loaded into the SAS session in a single step without a dependency on the SASAUTOS option.

In addition to user callable macros for the three major modules, there are several other user callable utility macros. These are required for functionalities that must be performed outside of the major macros, e.g. in a DATA step or PROC FORMAT that is user defined.

INITIALIZATION OF THE APPLICATION

An initialization macro must be called prior to using any of the other application's macros. It resides in our standard autocall macro directory, and is available to all SAS users and jobs. The initialization macro performs the following tasks:

- It %includes the system file of macros to load them into the SAS session and make them available in the SAS program. The location of RhoTables system is parameterized, which allows using an older version on a case by case basis, or delivering and executing RhoTables on a different platform without requiring a predefined location.
- It defines the connection to our Project Tracker Oracle database when used, or the location of the Project Tracker database when exported to SAS data sets.
- It defines the location of a RhoTables work directory for storing metadata created by the application.
- It defines a handful of global macro variables used across the application (directory locations, a persistent return code, etc.)

This approach has worked well and has been robust. It has allowed us to install and use the application on PC workstations, a Windows based SAS Grid, and on Unix/Linux with minimal modifications or issues. Further, there have been a couple of occasions when modifications were required for very special circumstances that we did not want to incorporate into the production system. The design allowed us to easily create and store a "special" version that was only used by a single project.

The approach of using an initialization macro that %includes the macros has an additional benefit of simplifying the testing of updates. In a test program, the initialization macro is called, loading the production macros, followed by %includes of the modified macros from the development area, which overwrites the production versions with the modified versions to be tested. This avoids the need to generate a new package for every iteration of testing.

STRUCTURE OF THE MACROS

The following principles that affect the overall structure of the system's macros should also be followed. These are not generally applicable to writing any SAS macro, but specifically apply to larger macro applications developed in a controlled environment.

- Individual macros perform specific and defined tasks.
- For an integrated system it is often useful to have a larger number of smaller macros each doing less, than having fewer larger macros doing more. This is mitigated by several factors including macro variable scope, DATA step processing (don't split a DATA step into two macros), etc.

  In contrast to a system, the more typical singular SAS macro may be rather large at times to avoid having to trace and maintain several macros where one will do. But once you cross the line into an integrated system, the rule flips and smaller becomes better, in general and within reason.
- All nested macro calls return control to the calling macro. Always maintain a straight path in with the identical path back out, e.g. all paths begin in and return to one macro, %RhoTables4.

  Some recommend limiting nested macros, or the depth of the nested levels. However, if properly designed and executed, a longer path of simpler macros may be less complex to follow than conditional processing in a larger macro. RhoTables goes up to 5 levels deep with nested macro calls.

- Always use a separate macro to perform a specific task in multiple places, even if only two. For example:
  - One macro writes all error messages and notes to the log.
  - One macro performs basic checks for valid parameter values.
  - One macro writes a solid horizontal line to the output.

NAMING CONVENTIONS

Names are important, but are often given little thought or consideration. It is important that the naming conventions for user called macros, global macro variables, data sets, etc. are meaningful, consistent, and unique to the application.

The natural human resistance to change elevates the importance of beginning with a meaningful naming convention that is also flexible and appropriate for future enhancements. Anticipating potential conflicts with other tools is also essential. Spending more thought and time in the beginning to anticipate potential enhancements and conflicts will avoid issues in the future. What you start with is usually what you are stuck with.

There are three major functional parts to RhoTables, with three corresponding user called macros:

- %RhoTables4 generates displays
- %RhoTables4Validate validates the displays
- %RhoTables4Stack stacks multiple displays into a single file

In hindsight, I would have named the macros %rt4Display, %rt4Validate, and %rt4Stack for brevity and consistency; however, when users were offered the option to change the naming conventions for version 4, that option, as well as others, was soundly rejected.

All other macros in the system are named %rt4xxxx, where xxxx is descriptive of its function. e.g. the initialization macro is named %rt4init.

Similarly, all global macro variables, temporary data sets, and formats created by RhoTables begin with the "rt4" prefix. Although not guaranteed to be unique, it is unlikely to be used in other macros and users quickly associate the convention with the application and avoid it in other uses.

I do not adhere to the practice of using underbars in an attempt to ensure unique names--it is such a widespread practice that it is more likely to create conflicts than avoid them.

The prefix convention was implemented religiously with version 3. This greatly facilitated updating SAS programs from version 3 to version 4, since for many programs it was a straightforward global search and replace of "rt3" with "rt4". This was something we got right from the start. Users learn quickly and know not to use the rt4 (or rt3) prefix for macro variable, data set, or other names—it's a simple rule that's easy to remember, and reliable so long as the consistency is maintained.

PROGRAMMING CONVENTIONS

Again, the rules and advice for programming practices may be rather different for a large system of macros such as RhoTables and the more typical singular SAS macro creation. The idea is to modularize both application functionalities and programming tasks as much as possible, to the extent it's beneficial. Therefore, some of what follows may contradict advice, or even admonitions, you have heard before.

It is essential to adhere to agreed upon programming conventions. Consistency is extremely important to being able to efficiently maintain the system over time. Specific recommendations include:

- Use consistent style elements such as indentation of `do;` constructs, capitalization, commenting, etc. The conventions used are much less important than being consistent in their use. When working with others be willing to compromise, even if it means diverging from your favorite habits.

- Comment every DATA step, PROC, do loop, and any logical section of code. Be religious, even when it seems redundant, boring, and annoying. Even if only you are responsible for maintaining the code, what is obvious to you when you write it will probably not be obvious a year or two down the road. If you implement any technique or feature that is new to you, comment it.
- Explicitly declare all macro variables, local or global. Always, with no exceptions (well, only one).
  - Note that in RhoTables all global macro variables ever required anywhere in the application are declared in the initialization macro. This makes them easy to identify and track.
  - The way SAS deals with macro variable scope is that nested macros will inherit local variables from the parent, if not declared as local in the called macro. It is best practice not to rely on this behavior, however, and to pass necessary variables as arguments to the child macro. For example:

    ```
    %let varx = 35;
    %rt4ChildMacro( varx=&varx );
    ```

    As an argument, `varx` will be local to `%rt4ChildMacro` as a distinct variable, which avoids inadvertent conflicts, but maintains consistency and readability.

    When passing a value from one macro to another, I prefer to use the same name. Typically, the value is used but not intentionally modified in the child macro, and this technique allows you to read individual macros knowing that it's the same value, while avoiding debugging nightmares without confidence in the macro variables' scopes, i.e. everything's local.
- The exception to the above rule is when the child macro must modify a variable in the parent macro. In general, it is best to avoid modifying the value of a macro variable that is used outside the macro, if possible. When required, however, there are two techniques to use:

  The preferred method is to pass the macro variable name (not the value) as an argument. For example:

  ```
  %macro rt4ChildMacro( parentVar=varx );
     …
     %if &condition = true %then %let &parentVar = newvalue;
     …
  %mend;
  ```

  This allows the child macro to be generically used without assumptions about the variable name and scope. *You do not want the child macro to create requirements and assumptions for the parent macro.*
- Do not use indirect macro variable references beyond one level. That is, limit multiple ampersands to three. Additionally, creating a temporary macro variable for the indirect reference can keep a macro more readable, especially if referenced multiple times.

  ```
  %macro rt4ChildMacro( parentVar=varx );
     ...
     %local parentValue;
     %let parentValue = &&&parentVar;

     /* all code now uses &parentValue instead of &&&parentVar */
     ...
  %mend;
  ```
- Quote everything (i.e. all macro references), until you know it contains a discrete controlled value (e.g. YES or NO).
  - You need to understand at least the basics of macro quoting.

- When testing arguments or other parameters for valid values, quote them, even when the valid values do not require quoting. Invalid values that a user may specify might contain characters that require quoting in order to avoid confusing SAS macro errors.

- Conversely, don't quote variables after you've tested them for valid values, except for strings and such that may contain special characters. Unnecessary quoting only creates more confusing and harder to read code.
- Use macros to compartmentalize code, simplifying logic and making the code in the calling macro more readable. For example:

  ```
  %if &WordTable = YES %then %rt4RenderRtfTbl( BDY );
  %else                      %rt4RenderRtfTxt( BDY );
  ```

  is much easy to follow than `%do; %end;` constructs with 500+ lines of code in each. Further, it is often easier to compare the difference between two files, than to compare code in two separate parts of the same file. In this case, the difference between rendering the output in a Word table versus rendering it as plain text.
- Similarly, you might replace a `%do;` loop enclosing 200 lines of code with a macro and 3 lines of code, simply to make it more readable. For example:

  ```
  %do I=1 %to &NCol;
    %rt4RenderCol(&I)
  %end;
  ```

- The judicious use of `%GOTO` simplifies the code and makes it easier to follow. In RhoTables:
  - All individual macros have a `%ENDMAC:` label at the end. Any condition that ends a macro's execution utilizes a `%GOTO ENDMAC;` statement. This is most commonly used when trapping errors, but is also useful when the remainder of the macro is not executed under normal conditions.
  - `%GOTO`s can also simplify complicated logic and replace deep nesting of DO conditions by skipping a whole section of code. However, `%GOTO` is only used to jump forward to skip a section of code, never to go backward.
- Follow SAS conventions whenever possible, particularly for arguments, parameters, and specifications. For example:
  - Data= for input data set
  - Out= for output data set
  - Where= for subsetting clause
  - etc.

  Ensure, however, that the parameters behave in the same manner as SAS.

  e.g. Data= may be either a 1 or 2 level data set name and _LAST_ is the default value.

  In some cases, the macro parameter may be more limited than the native SAS feature, but it should be consistent with SAS, given any inherent limitations of the application. For instance, it would be great if data set options are supported as part of the Data= argument, but not critical that they are.
- Do not, on the other hand, use a SAS name but with an application specific meaning or behavior. Keep it consistent with general SAS behavior.

SYSTEMATIC AND INTEGRATED ERROR CHECKING AND REPORTING

The general idea is that you don't want to rely on SAS errors to report an issue back to the user. We all know how obscure and confusing that can be, particularly when using macros. The goal is to terminate as cleanly as possible without other confusing SAS messages. Instead you want to relay application pertinent information back to the user, specifically (a) what went wrong; and (b) how to fix it in the context of the application.

The following rules guide the extensive checking of parameters and specifications:

- A single macro is used to write all messages (errors, warnings, notes) to the log for consistency.
- Every error, warning, or note generated by the macro system contains its name to identify the source and so it is not confused with a SAS generated message. e.g.

  ```
  ERROR: from %RhoTables4: Invalid value (Y) to the HdrTopLine= argument.
         It must be Yes or No.
  ```
- All parameters are verified for valid values.
  - Discrete list of valid values
  - Range for numeric values
  - Integer value
  - Real number
  - Date, time, or datetime specification
  - Data set name exists
  - Variable name exists in the data set
  - Expected variable type, numeric or character
  - Existing format name or specification
  - Directory path or folder
- All parameters are verified for consistency with one another where there are dependencies. This may never be 100% realized given the number of possible combinations, but the more complete the better.
- Trap all SAS errors possible and translate them into meaningful messages in the context of application. These can be difficult to anticipate and in the case of RhoTables error checking has greatly increased over time as creative users discover new interactions that were not anticipated.
- All error conditions are returned to the main calling macro so that there is a single end point to the application execution.
  - Error traps in any macro have a `%GOTO %ENDMAC;` (with a `%ENDMAC:` label at the end of the macro).
  - Nested macro calls are followed by a test of the persistent error flag. e.g. `%IF &RC ^= 0 %THEN %GOTO ENDMAC;` This causes nested macros to back out cleanly without continuing execution.
  - The controlling macro (e.g. %RhoTables4) tests the persistent error flag, and submits the `%abort;` statement which puts SAS into its normal error state, causing it to print the generic SAS error message at the bottom of the log.

DEBUGGING

One feature that has been indispensable is that the main macros contain a single unnamed argument that is used to run in debug mode and control the amount of output written to the SAS log. This is intended for developer use only. Possible values are:

OFF	No output to the log. Used only once when RhoTables calls itself.
INFO	Default. Writes parameters to the log.

NOTES	Prints SAS Notes from data steps. Includes INFO.
MPRINT	Turns MPRINT on. Includes NOTES and INFO.
DEBUG	Prints specific debugging messages and values programmed in the macros. All temporary data sets and output are saved. Includes MPRINT, NOTES, and INFO.

In Debug mode, all individual macros will at a minimum write a message to the log when they begin and when they end execution. In addition, the %PUT statement is widely used to identify locations in the code and to write pertinent macro variable values to the log.

In my opinion, using SYMBOLGEN and MLOGIC becomes impractical for this size of application. Instead use the debug flag to conditionally write macro values and code location labels to the log. This approach takes some effort to effectively implement, but is much more useful, and potentially essential, for identifying problems down the road.

THE DEVELOPMENT TEAM

Over the development of RhoTables there has been a consistent team of one experienced SAS programmer with extensive macro experience and some limited system development experience in other languages and environments. It is essential that developers have a good grasp of macro quoting issues, macro variable scopes, and nested macros.

The most difficult resourcing issue was for validation. This is a lot of work that is easy to underestimate and under value. Writing test cases and functional requirements require knowledge of SAS and the programs in which the macro will be used. We found it difficult to have non-SAS programmers perform the actual validation tasks, and in the end had a second experienced SAS programmer designated as the main validation programmer, with help from several other statistical programmers as resourcing allowed. This was a continuous resourcing challenge, since SAS programming positions are largely dedicated to full-time production work.

The software validation specialists utilized for version 4 were invaluable in developing and maintaining the required documentation for audits and bid defenses. Although having virtually no knowledge of SAS, they guided the validation process and dealt with the more bureaucratic aspects of the project.

Since RhoTables has been an application that has grown and matured over 14 years, our staffing and process has worked reasonably well for the most part. However, if we began with version 4 from a fresh start, significantly more project planning, resourcing, and coordination would be required up front.

CONCLUSION

Crossing the line from a single SAS macro to a system of macros requires careful consideration involving the validation process, code versioning, macro structure, programming conventions and techniques, and resourcing issues that may be significantly more involved as well as significantly different from typical daily macro writing. Planning and attention to these details before hand will help to ensure a successful development project and application.

CONTACT INFORMATION

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

John Ingersoll
Rho, Inc.
john_ingersoll@rhoworld.com