Paper 8-28

Fast and Easy Ways To Advance On Your Beginning SAS® Coworkers!

Rick M. Mitchell, Westat, Rockville, MD

ABSTRACT

This paper will provide the SAS programmer with advanced methods of reshaping one's approach to problem solving. With the abundance of options available in SAS, programmers can often rely too much on the defaults and thus hinder their growth beyond the beginning level. The primary concept that will be addressed in this paper is what a programmer must do to go beyond limiting oneself to an individual SAS tool in order to reach a more advanced programming level. The improvement of processes, the incorporation of nifty macros, and a focus on how to better present information, are all ingredients that are essential to a programmer who wants to go the extra mile. For programmers that have already achieved an advanced status, beware. There is plenty of room at the top as one discovers that there are fast and easy ways to advance on your beginning SAS coworkers!

INTRODUCTION

A major focus of this paper will be on **process improvement** and how multiple concepts may be utilized to advance one's work. Regardless of how advanced work may seem, there is often no limit to how far a program can be advanced. While a programmer may be considered "advanced" in only SAS/STAT, only SAS/GRAPH, or only in the area of macros, an even more advanced programmer is able to apply all of these tools and others to an overall process. SAS offers a huge warehouse with a variety of tools that often meet a programmer's minimal requirements. However, it is the advanced programmer who is able to force SAS to achieve the optimal look and feel that is desired or needed.

To continue the author's tradition of providing readers with "helpful" approaches that may deviate from the standards, "**Fast and Easy Tips**" are again provided for those readers who are looking for quick success or are perhaps just bored and looking for some fun and adventure in their lives. These tips, however, should be used at the programmer's extreme discretion, and the author claims no responsibility for their ultimate consequences!

OVERVIEW

This paper will discuss who fits the mold of an advanced programmer and how one might take his or her already "advanced" code to even higher levels by maximizing the use of multiple SAS tools. Potential areas where an advanced SAS programmer may be even more successful are discussed through the following topics:

- The advanced programmer
- Advancing "simple" code
- Advancing a process
- Advancing ODS output/graphics
- Advancing SAS/GRAPH
- Advancing secrets

While the SAS world is comprised of programmers with varying skill levels across many different disciplines, there are endless possibilities and opportunities for one to apply these concepts to everyday work.

THE ADVANCED PROGRAMMER

Fast and Easy Tip #1: Add 5 more years of experience to your resume!

Before proceeding further, it is important to discuss the mold that encompasses the advanced programmer. Because SAS offers such a wide variety of powerful and flexible tools, it is often possible for the beginning and advanced programmers to achieve the same or similar results. Therefore, it can be difficult to differentiate between the beginner programmer and the advanced programmer – at least on the surface.

What then distinguishes one from the other and at what point is one able to say that he or she has gone over the line into the advanced world of SAS programming? A beginning programmer may view the workplace as competitive when it is determined who gets the "challenging" tasks and who gets the "crumbs." However, some advanced programmers may view the "crumb" as the challenging task. In fact, just how far a task is taken does not necessarily depend on the request, but more on the programmer and how far the programmer can take the task. It is the truly advanced programmer who can make a mountain out of a crumb! Let us review the following questions to look further at these issues:

- What is a beginning programmer?
- What is an advanced programmer?
- Does use of a SAS component such as IML make a programmer advanced?
- What is opportunity and when does it occur?

Only after one fully understands these questions, may one begin to hold a firmer grasp on what makes an advanced programmer tick. Beginners may understand what direction to go in, while advanced programmers may see how to go even farther in that direction.

What is a Beginning Programmer?

Fast and Easy Tip #2: Blame mistakes on junior SAS programmers to ensure that they stay junior!

A beginning SAS programmer is one with either no experience, some experience, or even many years of experience covering the basics. These basics may often be limited to only one or two specific components of SAS. Years of experience are not always in sync with one's skill level. For example, a programmer who has worked for 5 years in SAS could still be considered a beginning programmer if this individual continued to only utilize the basics and never took (or never had the opportunity to take) advantage of additional tools. One who shies away from PROC TABULATE and hovers safely over multiple PROC FREQs may be considered a beginner. One who may be considered advanced in data manipulation but who prefers to continually export data to other graphics packages rather than learning the fine details of SAS/GRAPH might also be considered a beginner in the graphics area. When met with a challenge that cannot be met easily with SAS, a beginning programmer may be heard saying "Sorry, SAS does not do this."

What is an Advanced Programmer?

Fast and Easy Tip #3: Give hot job tips to threatening coworkers who infringe on your territory!

A programmer who may be considered advanced is quite simply one who is able to "wow" everyone. Regardless of the limitations of the SAS defaults, an advanced programmer is able to force the software to do absolutely anything. Difficult problems are rethought and reshaped such that the advanced programmer is constantly thinking how he or she might accomplish the task. Additional routes outside of the norm and odd coding schemes may be necessary to get SAS to do exactly what one wants. To the advanced programmer, this is everyday life in terms of going the extra mile, solving problems independently, and as a bonus, often giving something more than was originally requested. Being an advanced programmer is not necessarily knowing more complex code, but in a way, knowing where to go and what tools to take advantage of to successfully complete a task. Some concepts that an advanced programmer may use would take beginning programmers eons to figure out on their own. Advanced programmers are not merely those who can type the "correct" code the quickest, but more in how far one can take the task to fully utilize all of the tools that are provided by SAS.

Does Use of a SAS Component Such as IML Make a Programmer Advanced?

Fast and Easy Tip #4: Let the SAS/IML programmer think that he or she is doing important work while you do the REAL advanced work!

Most SAS programmers have never and will never use SAS/IML. It is a component of SAS geared primarily toward those heavy into mathematical and/or statistical areas where complex formulas are used involving calculations across various matrices of different sizes, with data streaming across rows, columns, and diagonals. Yuk! But, knowing how to use SAS/IML does not necessarily make one an advanced programmer, nor is one classified as a beginner programmer if he or she has never used SAS/IML. An optimal scenario would be if one was lucky enough to have an organization with multiple resources where a SAS/IML programmer could be utilized for the mathematical component and another programmer could be used to put all of the pieces together. The key here is how one would feed information into a component of SAS such as IML (or any other specialized SAS module) and how one would extract this information and present it. It is this process that is critical to the success of the advanced programmer.

What is Opportunity and When Does it Occur?

Fast and Easy Tip #5: Always take the most challenging tasks and leave the crumbs for others!

A large part of becoming an advanced programmer is knowing what an opportunity is, and knowing when it presents itself. To some programmers, opportunity may be the chance to work on something exciting that has been handed to them personally. This programmer might be approached with "Here's a great task. I want you to have the opportunity to work on it." But, could opportunity come along before the "big picture" is so apparent to everyone? One may never know what gold may be created out of a crumb without one taking the initiative and going into every task with an open mind. Have you ever heard one of your coworkers say "Why did Elmo get to take the lead on that task?" Perhaps that same coworker had turned down the opportunity a year earlier because the project was going nowhere and it just didn't seem fun enough. Most tasks have endless possibilities for advancement that are not always so clear at the start, and it is the advanced programmer who can often drive a task farther than all of the other programmers.

ADVANCING "SIMPLE" CODE

Fast and Easy Tip #6: Stall the requestor of a task until the next version of SAS comes out!

Let us begin to look at some examples of beginner code stretching out to advanced code, and then advancing on that a little further. Limitations in the flexibility of PROC FREQ can affect just how good one's output will look, and in the case of this example, one is limited to the length of variable labels. While SAS Version 8 amazed programmers with the opportunity to go beyond 8 character format labels in the rows of cross tabulations, the recently new 16 character wrap around feature has its limitations as well. Beginner programmers may stop and consider their job completed with the output shown in Figure 1, although "Less than monthly" and "Daily/almost daily" are sloppily wrapped. However, advanced programmers would realize that there's much more work to be done. It is not necessarily the code that is advanced, but the fact that the programmer knows that there is another angle to solving the problem!

```
value patfreq
   0='Never             '
   1='Monthly +         '
   2='Less than monthly '
   3='Weekly            '
   4='Daily/almost daily';
_______________________________________

patfreq(Pats on the back)
                  empmood(Employee Mood)
Frequency        |Happy   |Sad     |  Total
_________________|________|________|
                 |        |        |
Never            |       5|      91|     96
_________________|________|________|
                 |        |        |
Monthly +        |       9|      32|     41
_________________|________|________|
                 |        |        |
Less than monthl |      15|      11|     26
y                |        |        |
_________________|________|________|
                 |        |        |
Weekly           |      23|       3|     26
_________________|________|________|
                 |        |        |
Daily/almost dai |      57|       2|     59
ly               |        |        |
_________________|________|________|
Total                  109      139     248
```

Figure 1. Beginner Crosstab in PROC FREQ

A quick solution to this problem might simply be to utilize ODS RTF since the default for such an output would provide the additional column space needed (See Figure 2).

Table of patfreq by empmood			
patfreq(Pats on the back)	empmood(Employee Mood)		
Frequency	Happy	Sad	Total
Never	5	91	96
Monthly +	9	32	41
Less than monthly	15	11	26
Weekly	23	3	26
Daily/almost daily	57	2	59
Total	109	139	248

Figure 2. Default ODS RTF – Without Format Mods

But, let's say that using ODS was not an option for this task. What might one do to "pretty up" the output by just using the immediate "basic" tools? The "advanced" approach that one might take would be to simply modify the format labels such that the formats would break appropriately for a given report as shown in Figure 3. While this solution seems quite simple, most beginner programmers may not think of this alternative approach. Again, having the ability to consider other solutions outside of the typical boundaries is what helps shape an advanced programmer. While there is no magical option to give the programmer immediately what he or she wants, this does not mean that it cannot be accomplished in SAS.

```
value patfreq
   0='Never             '
   1='Monthly +         '
   2='Less than      monthly'
   3='Weekly            '
   4='Daily/almost    daily  ';
_______________________________________

patfreq(Pats on the back)
                  empmood(Employee Mood)
Frequency        |Happy   |Sad     |  Total
_________________|________|________|
                 |        |        |
Never            |       5|      91|     96
_________________|________|________|
                 |        |        |
Monthly +        |       9|      32|     41
_________________|________|________|
                 |        |        |
Less than        |      15|      11|     26
monthly          |        |        |
_________________|________|________|
                 |        |        |
Weekly           |      23|       3|     26
_________________|________|________|
                 |        |        |
Daily/almost     |      57|       2|     59
daily            |        |        |
_________________|________|________|
Total                  109      139     248
```

Figure 3. "Advanced" Crosstab in PROC FREQ

While the frequency in Figure 1 would have nicely defaulted to the ODS RTF output in Figure 2 with no modification of the format being needed, the new formats in Figure 3 would produce undesirable results in the default for ODS RTF output as shown in Figure 4. To further complicate matters, Figure 4 would have looked identical to Figure 2 had ODS HTML output been created. Therefore, it is very important for a programmer to fully understand the consequences of each programming decision. What works for one task may not work for another. The advanced programmer must be aware of these idiosyncrasies when pooling together SAS tools.

Table of patfreq by empmood			
patfreq(Pats on the back)	empmood(Employee Mood)		
Frequency	Happy	Sad	Total
Never	5	91	96
Monthly +	9	32	41
Less than monthly	15	11	26
Weekly	23	3	26
Daily/almost daily	57	2	59
Total	109	139	248

Figure 4. Default ODS RTF – With Format Mods

ADVANCING A PROCESS

Fast and Easy Tip #7: Employ programmers who can type lots of code VERY fast!

Process improvement is key to an advanced programmer. While many problems can be addressed by any programmer using the most basic tools, larger requests may sometimes be problematic for the beginning programmer. For such work, it is essential for the programmer to improve on the time and efficiency in which a program is run unless of course one chooses to type many times more code than is actually needed. The final deliverable (the output) must be carefully considered as well. Regardless of how good a programmer's code is considered to be, there is always most likely some aspect of it that can be taken a step farther. Whether this extra step or two is obvious or not most often depends on the skill level of the programmer. A beginning programmer may consider the day's work completed, while an advanced programmer will continue to push, pull, kick, thrust, or do whatever is necessary to get the code to process in the needed manner. Let us review a progression in code starting at a level at which we might expect a beginner to pursue.

- Task 1: Provide a crosstab using PROC FREQ of daysoff vs. empmood with chi-square statistics

This task (#1) is quite simple and even the most beginner programmer should be able to easily generate the code to complete the task. All of the necessary information was provided in the request and the resulting code would merely be what is shown in Figure 5.

```
proc freq data=sugidata;
   tables daysoff*empmood / chisq;
run;
```

Figure 5. Beginner Approach

A programmer who wanted to get a little more advanced might anticipate that further requests may be coming down the road and additional variable comparisons may be needed. Therefore, the programmer could build a simple macro to accommodate such requests where the macro would drive what variables would be compared as shown in Figure 6. If the programmer knew that a variety

```
%macro freqruns (var1=, var2=);
   proc freq data=sugidata;
      tables &var1 * &var2 / chisq;
   run;
%mend freqruns;

%freqruns(var1=daysoff, var2=empmood)
```

Figure 6. Beginning to Prepare

of crosstabs may be requested in the future, then this mechanism would allow for each input of 2 variables for each run. While this may not seem like a huge savings in code, consider the requestor who ends up asking for several hundred frequencies, all of which need to have some standard look.

- Task 2: Provide a cross tabulation using PROC FREQ of daysoff and paydecreases vs. empmood and provide chi square statistics as well as trend tests.

After receiving Task 2, the programmer may begin to realize that even though he or she had anticipated that additional variables may be requested, the programmer had assumed that only chi-square statistics would be run and this rule does not hold true. Now the requestor has also asked for trend tests, and the task will need to be further enhanced. Nonetheless, the programmer is able to very simply advance the code to perform this task by further automating the task with macros. By continuing to build on this macro, the programmer offers an increased number of choices where a wider variety of runs may be performed by simply specifying the appropriate variables and statistical tests as shown in Figure 7.

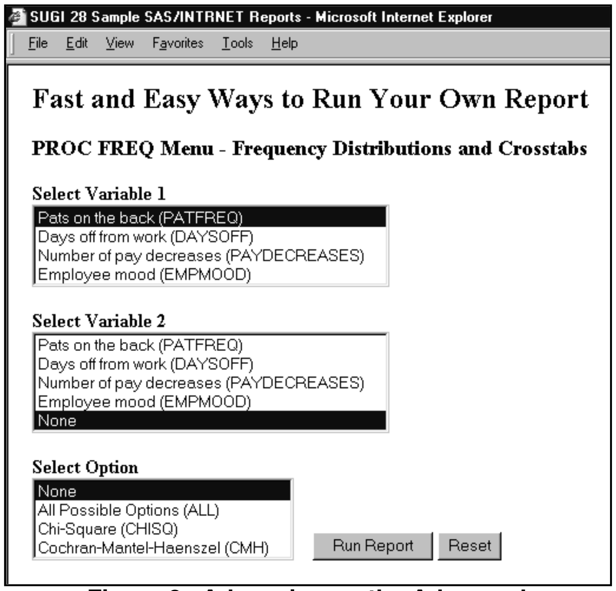
```
%macro freqruns(var1=, var2=, freqtest=);
   proc freq data=sugidata;
      tables &var1 * &var2 / &freqtest;
   run;
%mend freqruns;

%freqruns(var1=daysoff, var2=empmood,
   freqtest=chisq trend)
%freqruns(var1=paydecreases, var2=empmood,
   freqtest=chisq trend)
```

Figure 7. Preparing to Advance

These programming statements are presented in a simple manner, but one can perhaps imagine the potential applications for such an approach with more complex algorithms. A project that would require such programming flexibility because of the number of ever changing requests or more importantly, the massive number of requests, would certainly benefit from such an approach. This paper will not delve further into the detailed possibilities for enhancing such an approach, but one can imagine the potential uses for work dealing with the generation of many runs (e.g. logistic), model building, and routinely changing variables and options. While these previous examples have shown more advanced programming than was originally noted in Task 1, the intended audience of this paper is the advanced SAS programmer, so we will therefore use these examples as a foundation for advancing our code even farther.

- Task 3: Provide a mechanism for a non-programmer to run multiple SAS procedures on multiple variables using multiple tests.

With the increasingly high number of tools provided by SAS, programmers are now able to go beyond the basics, beyond automation of just simple macros, into an entirely different world. It is this new world that allows programmers to meet Task 3 by setting up simple menus for co-workers or even clients to perform various runs at their own convenience without needing to generate official programming requests for run after run. It is this issue that we will pursue as one may ask, what could one do to be even MORE advanced with one's code? So, back to the purpose of this paper – identify all of the tools that are available and incorporate them into a process that is easy to use and to understand. The example shown in Figure 8 does exactly this. One is able to go beyond the


SUGI 28 Sample SAS/INTRNET Reports - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Fast and Easy Ways to Run Your Own Report

PROC FREQ Menu - Frequency Distributions and Crosstabs

Select Variable 1

Pats on the back (PATFREQ)
Days off from work (DAYSOFF)
Number of pay decreases (PAYDECREASES)
Employee mood (EMPMOOD)

Select Variable 2

Pats on the back (PATFREQ)
Days off from work (DAYSOFF)
Number of pay decreases (PAYDECREASES)
Employee mood (EMPMOOD)
None

Select Option

None
All Possible Options (ALL)
Chi-Square (CHISQ)
Cochran-Mantel-Haenszel (CMH)

Figure 8. Advancing on the Advanced

traditional nifty macros and actually allow the end users to do some individual frequency or crosstab runs of their own. In this PROC FREQ menu, one is able to select up to two variables as well as a single PROC FREQ option within the TABLE statement. Imagine the look on the requestors' faces when they are presented with a simple, menu-driven set of HTML screens that allow them to do some exploratory work on their own without bothering with all of those official request forms for programming!

While this is a very basic example, one might envision the endless possibilities for code advancement here as this simple example could be greatly expanded. SAS/IntrNet was used for this example, but one could have also taken advantage of SAS Enterprise Guide or SAS Analyst depending on the needs of the project. SAS/IntrNet allows one to gear a menu driven system toward a specific project or specific client whose needs may be much different than another client. One client may be a statistician with some programming experience while another client, or even coworker, may have absolutely no background in this area and would need more specific guidance. Of course, with such a case, one would want to be very cautious regarding just what information could be extracted so as to not have one of those "clueless" employees misinterpret the data. By utilizing SAS/IntrNet, one would have the following advantages:

- The user does not need to have SAS software. So, one could set up a menu on a network for internal use by coworkers, or put on a server to be utilized via a web page by anyone in the world..
- Time and expense could be significantly cut back if one were able to do ad-hoc queries or much of the preliminary work oneself.

Some of the basic code used to generate the example in Figure 8, is shown below in Figure 9. Note that these HTML statements are quite basic and that there is considerable room for elaboration.

```
<h2>Fast and Easy Ways to Run Your Own Report</h2>
<h3>PROC FREQ Menu - Frequency Distributions and Crosstabs</h3>

<b>Select Variable 1</b><br>

<select name=report2 size=4>
<option value="*patfreq" selected>Pats on the back (PATFREQ)
<option value="*daysoff">Days off from work (DAYSOFF)
<option value="*paydecreases">Number of pay decreases
   (PAYDECREASES)
<option value="*empmood">Employee mood (EMPMOOD)
</select><br><br>

<b>Select Variable 2<br>

<select name=report3 size=5>
<option value="*patfreq">Pats on the back (PATFREQ)
<option value="*daysoff">Days off from work (DAYSOFF)
<option value="*paydecreases">Number of pay decreases
   (PAYDECREASES)
<option value="*empmood">Employee mood (EMPMOOD)
<option value=" " selected>None
</select><br><br>

<b>Select Option</b><br>
<select name=report4 size=4>
<option value=" " selected>None
<option value="trend">All Possible Options (ALL)
<option value="chisq">Chi-Square (CHISQ)
<option value="or">Cochran-Mantel-Haenszel (CMH)
</select>

<input type="HIDDEN" name="_PROGRAM"
   value="myfile.sugi.sas">
<input type="HIDDEN" name="_SERVICE" value="ab">
<input type=submit value="Run Report">
<input type=reset value="Reset""></p></a>
```

Figure 9. HTML Code For PROC FREQ Menu

In Figure 10 below, code is shown for the SAS program where various macros are called based on the information generated by the user from the menu system.

```
ods html body=_webout(dynamic title='Reports')
      path=&_tmpcat (url=&_replay) rs=none;
   proc freq data=all;
      tables &report2&report3 / &report4;
      title1 "Job Satisfaction Study";
   run;
ods html close;
```

Figure 10. HTML Code For Creation of Menu

This process ultimately results in output generated for the user with a few simple clicks of the mouse. The user can immediately save the output, or quickly go back to the menu and perform additional queries. In summary, the advancement of a process is really a project filled with endless possibilities. The advanced programmer is the one who can visualize all of these steps and determine which one may best fit a given project. For some projects, a basic PROC FREQ statement may do the trick and any further work would be overkill. But for others, there are few limits to one's ability to advance work not only for the programmer, but for the client as well.

ADVANCING ODS OUTPUT/GRAPHICS

Fast and Easy Tip #8: Sell your SAS code to junior coworkers in need of quick solutions!

ODS was a huge breakthrough for SAS. The introduction of this powerful tool allowed programmers to more easily retrieve important output information with much less programming. It does, however, still take some clever thinking to get exactly what one wants. One problem faced by many programmers is that getting a grasp of ODS output files can be difficult. Documentation by SAS may be insufficient for some programmers, and they may frequently need to rely on other users (e.g. SUGI, books by users, and coworkers) to work through difficult ODS issues.

Making Sense of the ODS Output Datasets

To provide more insight into such difficulties, let us take an example where data are generated and filtered through PROC GENMOD. To first understand the various ODS output files that are available, one must issue an ODS TRACE ON statement where the result is a listing of information within the SAS LOG of all of the available files. An example of one file is shown below in Figure 11,

```
Output Added:
-------------
Name:       ModelInfo
Label:      Model Information
Template:   Stat.Genmod.ModelInfo
Path:       Genmod.ModelInfo
```

Figure 11. Example of ODS Output Description

although there are actually six such files offered by PROC GENMOD. Note that the file information can be difficult to remember for just a single procedure not to mention all of the SAS procedures, especially if a user does not utilize these tools very often. The difficulty even grows as one looks at all of the Template descriptions that are provided by PROC GENMOD and realizes that detail is essential. In Figure 12, one will see that not only must one remember these extremely long output file names, but SAS has not yet perfected consistency among the names. These names seem to vary frequently between upper and lower case as shown in the listing of template names (e.g. “Stat.Genmod” vs. “stat.genmod”). If a programmer were to call on these names in some automated process, then this issue could affect its ability to run successfully.

```
Stat.Genmod.ModelInfo
Stat.Genmod.Classlevels
stat.genmod.ModelFit
Stat.Genmod.ConvergenceStatus
stat.genmod.parameterestimates
Stat.Genmod.Obstats
```

Figure 12. ODS Templates For PROC GENMOD

It is this difficulty that leads us to our next “advanced” topic. How can one avoid the hassle of having to figure out what the output name is called, and then rerunning the procedure to collect the desired information into the appropriate ODS output file(s)? The solution is a macro called OdsInfo that contains 3 distinct modules. The first module is shown in Figure 13 where ODS TRACE is turned on with a LISTING, and an ASCII file is created to capture everything generated in the SAS Log.

```
%macro OdsInfo (module=, masterds=, procname=);
   %if &module=capture %then %do;
      proc printto print='c:\testout.txt' new;
      ods trace on/listing;
   %end;
```

Figure 13. Module 1 Code - Capture SAS Log

This first module is implemented by calling the macro with the module variable set to “capture.” The system would then be ready for a user to run any SAS procedure such as PROC GENMOD as is run in Figure 14.

```
%OdsInfo(module=capture)

proc genmod data=examp;
   class patfreq daysoff;
   model empmood = patfreq daysoff /
      dist=poisson link=log obstats;
```

Figure 14. Module 1 Call – Capture SAS Log

The second module is shown in Figure 15 where the ASCII capture file is closed and a dataset is created that contains ODS output information on whatever SAS procedures have been run. Note that this dataset is able to automatically identify the procedure that was run, and that only new ODS file names are added to the master file of ODS file information.

```
%else %if &module=collect %then %do;
   proc printto; run;
   data newinfo(drop=oread1 oread2);
      retain oname olabel otemp opath;
      infile 'c:\testout.txt' truncover;
      input oread1 $ 1-5 oread2 $ 13-50;
      if oread1='Name:' then oname=oread2;
      else if oread1='Label' then olabel=oread2;
      else if oread1='Templ' then otemp=oread2;
      else if oread1='Path:' then do;
         opath=oread2;
         do i=1 to 20;
            if substr(opath,i,1)='.' then i=20;
            else if i=1 then procname =
               substr(opath,i,1);
            else procname=left(trim(procname))
               || (substr(opath,i,1));
            end;
            output;
         end;
    proc sort data=&masterds; by opath; run;
    proc sort data=newinfo; by opath; run;
    data &masterds;
       merge &masterds(in=a) newinfo(in=b);
        by opath; run;
%end;
```

Figure 15. Module 2 Code – Collect SAS Log

This second module is implemented by calling the macro with the module variable set to “collect” and identifying the name of the master ODS collection file as shown in Figure 16.

```
%OdsInfo(module=collect, masterds=mylib.odsinfo)
```

Figure 16. Module 2 Call – Collect SAS Log

Note that the program will compare a new file of ODS names to the master file of names used so far by the program. While the ODS names only need to be captured and collected the first time that the procedure is used, the program will continue to flow in the same manner regardless of the number of times that it is run. The program will only add new file names to the master file. One caveat to this process would be that depending on the statements and options that are used at the time of the first run, it is possible that all output files will not necessarily be generated. For example, running PROC FREQ with a CHISQ option would generate 2 ODS output files (CHISQ and CROSSTABFREQS) while the use of the ALL option would produce 4 files (CHISQ, CROSSTABFREQS, CMH, and MEASURES).

The third and final module, set to “open” (see Figure 17) will read the master ODS collection file that contains ODS file names for a given procedure and generate the necessary ODS OUTPUT statements (see Figure 18).

```
   %else %if &module=open %then %do;
      data _null_;
        set &masterds;
        where procname="&procname";
       call execute("ods output " || trim(oname)
          || "=" || trim(oname) || ";");
      run;
   %end;
%mend OdsInfo;

%OdsInfo(module=open, masterds=mylib.odsinfo,
   procname=Genmod)
```

Figure 17. Module 3 Code/Call – Open ODS Files

```
NOTE: There were 6 observations read from the
data set WORK.ODSFILES.

MPRINT(ODSINFO): ods output ModelInfo=ModelInfo;
.
.
.
MPRINT(ODSINFO): ods output ObStats=ObStats ;
```

Figure 18. Automatic Call For ODS Output Files

With all of this automated processing now behind us, one may now run a SAS procedure whose ODS information has been captured, collected, and opened and the corresponding SAS datasets will automatically be created (see Figure 19). Note that another caveat with this process is that if one wanted to call PROC GENMOD several times, then the ODS output files could potentially overwrite each other. This limitation could surely be overcome with a little additional code by the adventurous programmer.

```
proc genmod data=examp;
   class patfreq daysoff;
   model empmood = patfreq daysoff /
      dist=poisson link=log obstats;
run;

NOTE: Algorithm converged.
NOTE: The scale parameter was held fixed.
NOTE: The data set WORK.OBSTATS has 6
   observations and 17 variables.
NOTE: The data set WORK.PARAMETERESTIMATES has 7
   observations and 9 variables.
NOTE: The data set WORK.CONVERGENCESTATUS has 1
   observations and 2 variables.
NOTE: The data set WORK.MODELFIT has 5
   observations and 4 variables.
NOTE: The data set WORK.CLASSLEVELS has 2
   observations and 3 variables.
NOTE: The data set WORK.MODELINFO has 6
   observations and 3 variables.
```

Figure 19. Automatic Creation of ODS Output Files

One may not actually need some or any of these output datasets. Generating unneeded datasets may seem to be a waste to some, but in reality, the datasets are quite small. In this day and age, electronic data storage is cheap, and their quick availability may save time in the long run as one can avoid having to figure out how to access a given ODS output file. By not accepting ODS as it is sometimes awkwardly provided, one is able to advance on coworkers unwilling to twist SAS a bit here and there to force it to work for you.

Going Beyond the Basic ODS Graphics

Fast and Easy Tip #9: Delete SAS’s template directory from the network and store it on your personal hard drive!

Coming out in SAS Version 9.0 Production, SAS boasts additional graphics that are automatically included with ODS output for several of the SAS/STAT and SAS/ETS procedures. This STAT/GRAPH feature is a warm welcome to those programmers wanting to give statisticians and analysts a nice, quick look at the results without having to put forth any effort. These graphics are produced automatically within an ODS output destination (RTF, PDF, HTML, etc.) when including:

ODS GRAPHICS ON;

While on the surface this may seem quite wonderful, one must be cautious of those requestors who always seem to ask for a little bit more. In SAS Version 9.0, this is one big con of using STAT/GRAPH as one is not able to modify a graph that is produced and no code is provided. So, while these ODS graphics may be intended for a general look at the results, one might expect a more advanced programmer to head back to the ever so powerful tool, SAS/GRAPH to accomplish the same task.

Let us review an example of an autocorrelation graph produced with PROC ARIMA as shown in the SAS documentation, "Statistical Graphics Using ODS in Version 9 (Experimental)." Note that PROC ARIMA is a procedure within SAS/ETS that "predicts a value in a response time series as a linear combination of its own past values, past errors (also called shocks or innovations), and current and past values of other time series." The SAS statements in Figure 20 are taken directly from the documentation as well as the graphic that is automatically generated in ODS (see Figure 21).

```
proc arima data=airline;
   identify var=logair(1,12) nlag=15;
run;
```

Figure 20. Example Code From SAS Documentation

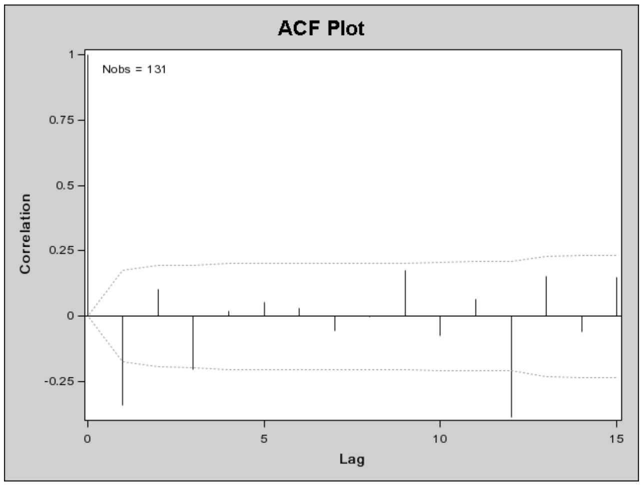


Figure 21. ODS With Stat/Graph

Perhaps this may be as much as one hopes to accomplish, but why not provide users with PROC GPLOT code within SAS/GRAPH? This would not only produce this exact same graph, but would also give the programmer much greater flexibility and control over the look of the output. For example, if one wanted to change an axis scale, use different lines or values, or even provide special annotations or notes, the STAT/GRAPH output would not currently allow this in SAS 9.0. Luckily, it sounds as if SAS has plans for future releases to include the mechanism of providing code, templates, and adding graphics to many other procedures. Unfortunately, these plans will not benefit the programmer who has work that is due by the close of business today!

So, one asks what an advanced programmer might do when taking a task a step farther, this offers a good example of such a case. As mentioned, this same graph could have been produced by first accessing PROC ARIMA's output data and utilizing the covariance output datasets as shown in Figure 22. Note that this is as simple as including a few words and then taking the extra step of setting up the confidence intervals of the mean (plus or minus 1.96 times the standard error).

```
proc arima data=airline;
   identify var=logair(1,12) nlag=15
      outcov=outcov;
run;
data outcov;
   set outcov;
   limit=1.96*stderr;   /* upper limit */
   output;
   limit=-1*limit;      /* lower limit */
   output;
run;
```

Figure 22. Creating a File For SAS/GRAPH

Note that the data that make up the graph are very straightforward as shown in the ODS autocorrelations table below in Figure 23 where the correlations and standard error values are used for plotting.

Autocorrelations			
Lag	**Covariance**	**Correlation**	**Std Error**
0	0.0020860	1.00000	0
1	-0.0007116	-.34112	0.087370
2	0.00021913	0.10505	0.097006
3	-0.0004217	-.20214	0.097870
4	0.00004456	0.02136	0.101007
5	0.00011610	0.05565	0.101042
6	0.00006426	0.03080	0.101275
7	-0.0001159	-.05558	0.101347
8	-1.5867E-6	-.00076	0.101579
9	0.00036791	0.17637	0.101579
10	-0.0001593	-.07636	0.103891
11	0.00013431	0.06438	0.104318
12	-0.0008065	-.38661	0.104621
13	0.00031624	0.15160	0.115011
14	-0.0001202	-.05761	0.116526
15	0.00031200	0.14957	0.116744

Figure 23. Example Output From PROC ARIMA

While the additional code to PROC ARIMA was miniscule, the author must admit that the additional code that would be needed to produce the same graph with PROC GPLOT would take some extra time. For the advanced programmer, this time may be small as the annotation dataset and plotting statements are very basic for the typical SAS/GRAPH user. Note that the code shown in Figure 24 is not necessarily complicated, but it is the advanced programmer that would know how to quickly and efficiently put such a process together. This code will provide a graph that looks very similar to the graph that was automatically produced with ODS. It is with this minimal code that a programmer may continue to enhance.

```
data anno;
   length function style color $ 8 text $ 25;
   function='label'; x=20.5; y=42;
   text='Nobs = 131'; style='swissb'; size=1.0;
   position='6'; output;

proc gplot data=outcov anno=anno;
   plot corr*lag limit*lag /
      overlay frame vaxis=axis1 haxis=axis2
      nolegend;
   axis1 label=(angle=90 rotate=0 font=swissb
                'Correlation')
         order=(-0.25 to 1.00 by 0.25)
         value=(font=swissb)
         minor=none offset=(7.6,2);
   axis2 label=(font=swissb 'Lag')
         order=(0 to 15 by 5)
         value=(font=swissb)
         minor=none offset=(1,1);
   title1 font=swissb h=5.0 'ACF Plot';
   symbol1 v= font=swissb h=1.0 interpol=needle
      line=1 w=2.0 mode=include;
   symbol2 v= font=swissb h=1.0 interpol=join
      line=33 w=2.0 mode=include;
```

Figure 24. PROC GPLOT Code

By utilizing this process, the advanced programmer may now have total control over the look and feel of the graph, as will future programmers that may inherit the code. Itis not the ODS that has given this new look to SAS, but it is the advanced programmer who once again has taken SAS beyond the limits.

ADVANCING SAS/GRAPH

Fast and Easy Tip #10: Bury a line in your program that will shut everything down after you quit your job such as,

```
if substr("&sysdate",6,2) > '03' then stop;
```

SAS/GRAPH can be difficult to learn, and it can be intimidating to beginning programmers. This may also hold true for advanced programmers who use SAS/GRAPH on only rare occasions. There are two primary reasons for this difficultly: (1) if one does not use the software routinely, frequent reference to documentation is most often needed, and (2) documentation provided by SAS may not always meet the needs of all programmers. However, despite the obstacles that one faces when trying to grasp these graphical programming skills, a mastery of SAS/GRAPH for even the most advanced SAS programmer will give one the edge over other advanced programmers. This edge comes in one being able to apply SAS/GRAPH to a process such that the challenge is not just in creating a graph, but in pooling multiple SAS resources together to solve a problem and present it to a wider audience. Let us take an example of a graphical presentation of odds ratios (Mitchell, 2000) as shown in Figure 25 (for detailed SAS statements that are used to generate this graph, see “Forcing SAS/GRAPH Software To Meet My Statistical Needs: A Graphical Presentation of Odds Ratios” in the SUGI 25 Proceedings).

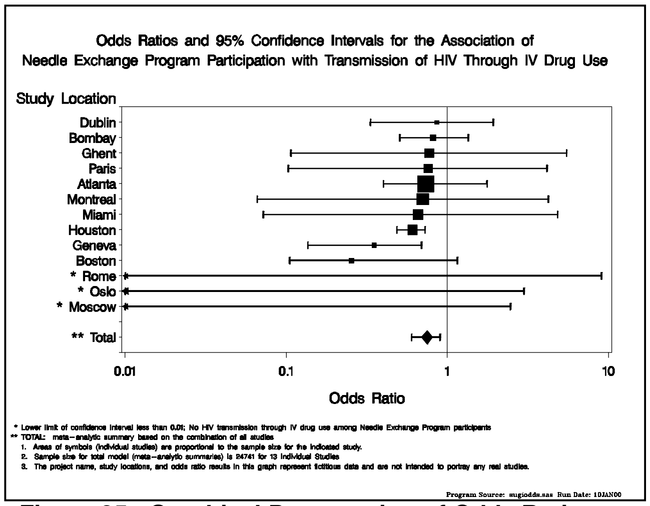


Figure 25. Graphical Presentation of Odds Ratios

This graph itself is not necessarily an advanced concept as it is actually very common throughout statistical journals where odds ratios are compared amongst multiple studies. Unfortunately for the beginning programmer, there is no magical PROC ODDSRATIO in SAS/GRAPH so the effort to create the graph was fairly challenging. What is so important about this graph is that it went far beyond what SAS offers the general user. This graph combined a variety of SAS tools and incorporated them into multiple processes to generate graphs for varying projects, studies, populations, and notes. The programmer ignored the limitations of SAS/GRAPH and presented information including the following:

- Automatic generation of SYMBOL statements based on the number of participating studies
- Automatic generation of shape sizes within SYMBOL statements to show population sizes
- Automatic connecting of multiple lines to form confidence intervals in a horizontal format (SAS/GRAPH provides only vertical)
- Automatic generation of value symbols for confidence intervals with special intervals (less than the lowest or highest axis value)
- Automatic generation of footnotes for special study exceptions

This graph can handle most datasets for any type of study. Its components utilize the DATA STEP, macros, PROC LOGISTIC in SAS/STAT, and PROC GPLOT in SAS/GRAPH. Each of these components by themselves are actually quite basic, but the combination of these components together in an automated manner is what makes the graph so unique. The effort that went into creating the graph has been far surpassed by its use. The graph has not only been utilized across multiple projects within the author’s workplace, but it has also been distributed and utilized by other programmers throughout the world!

ADVANCING SECRETS

Fast and Easy Tip #11: Create your own secret directories, programs, and libraries!

Not all ways of advancing on your SAS coworkers are technical in nature. Regardless of one's technical know-how, efficiency in programming, and nifty macro processing skills, an advanced programmer must often earn the trust and respect of coworkers in order to be successful. Let us consider 4 important rules that may also help a programmer advance:

- Spread the wealth
- Show me yours, and I'll show you mine
- Take responsibility
- Don't forget where you came from

By incorporating these rules into a programmer's everyday life, one may become a more complete advanced programmer.

Spread the Wealth

Fast and Easy Tip #12: Take credit when others aren't there to accept it!

With multiple programmers often collaborating on a project together, it is not so uncommon for one to be overlooked when praise is given. The insecure programmer may merely say "thanks" and ignore any contributions from coworkers. The acknowledgement of other team members who helped make something happen is imperative, regardless of the success or failure of a project. Spreading the wealth does not threaten the truly advanced programmer as coworkers' successes (perhaps even more than your own) should be embraced.

Show Me Yours, and I'll Show You Mine

Fast and Easy Tip #13: Never share a new tip or technique that you've learned!

On a project with multiple programmers, sharing programs and techniques can be very beneficial. One may sometimes be wary of showing one's program to others for two reasons: (1) programmers may be insecure about their work and worry that others will find mistakes or criticize their approach, or (2) programmers may worry that others will pick up on their nifty programming tricks. It can be reassuring to have a fresh set of eyes look something over that may have become a blur to you some 1,000 lines of code ago! Additionally, by being open with one's coworkers, one may further gain their trust and others may be more likely to come to you with neat new techniques that could save you lots of time and frustration in the future.

Take Responsibility

To gain long-term credibility, it is essential that one take full responsibility for the work that one produces. Honesty and openness will go a long way in getting others to trust your work, as well as being able to keep more junior SAS programmers under you in a motivated environment. If you make a mistake, admit it! Taking responsibility will not only make you feel good, but it will also ensure the integrity of your work with the client.

Don't Forget Where You Came From

All advanced programmers were beginners at some point in their lives. By keeping this in mind, an advanced programmer may better help beginning programmers by giving them much needed guidance and support (with patience!). The advanced programmer can be of valuable assistance to the beginning programmer who also hopes to quickly grow as a SAS programmer.

CONCLUSION

Regardless of the number of years of SAS programming experience that one may have under his or her belt, there is ALWAYS room for improvement. Even advanced programmers should be able to find some way to improve a process for either their own benefit, a team benefit, or their client's benefit. While a programmer may utilize advanced techniques within a given component of SAS, it is the advanced programmer who can successfully pool techniques across multiple components to advance code even farther than most can imagine.

REFERENCES

- Mitchell, R.M. (2000). Forcing SAS/GRAPH Software to Meet My Statistical Needs: A Graphical Presentation of Odds Ratios (Paper 167-25). *Proceedings of the 25th Annual SAS Users Group International (SUGI) Conference*.
- SAS Institute (2002). Statistical Graphics Using ODS in Version 9 (Experimental). *ODS Documentation*.

ACKNOWLEDGEMENTS

SAS, SAS/GRAPH, SAS/IML, SAS/INTRNET, SAS/STAT, and SAS/ETS are registered trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

DISCLAIMER: The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

CONTACT INFORMATION

Rick M. Mitchell
Westat
1650 Research Boulevard, WB 496
Rockville, MD 20850
(301) 251-4386 (voice)
(301) 738-8379 (fax)
RickMitchell@Westat.com