ABSTRACT

For SASR users needing capabilities over and above those provided by the reporting procedures this paper discusses the techniques for and the process of generating periodic production reports. The design, construction, and implementation phases of the report writing project are discussed as necessary prerequisites for acquiring a superior end product. The FILE and PUT statements, system options, and data step functions are demonstrated to control the placement of variable and constant data on the printed product. Additionally, the Macro Facility is explored as a method for generating periodic reports. Since most of the material discussed is well explained, along with examples, in SAS System documentation, the emphasis is on a conceptual understanding of the report writing process.

INTRODUCTION

Report writing with the SASR System can vary with regard to the level of difficulty and complexity. For the "quick and dirty" approach a simple SAS reporting procedure, such as the PRINT or MEANS procedure, may be perfectly adequate. Reports may also take on an "ad hoc" nature. These reports usually have one time or limited usage, but need more sophistication than a SAS procedure. The "periodic production report" is the report type of most concern to the serious SAS programmer. These reports will be of a consistent, known form and therefore worth the time required to construct them. They will also by necessity be useable by non-technical individuals. This tutorial will address the development of the latter report type, although the methods discussed are not limited to the periodic production report.

First and foremost the data to be reported on must be in the form of a SAS data set. This can be accomplished by either using an existing SAS data library, output from a statistical procedure, or a temporary creation from a raw input file. Once the data is in a reportable form, the complexity required of the reporting data step is wholly dependent upon what is needed by the recipient audience.

For those SAS programmers responsible for the development and support of the periodic production report, several important project phases must be addressed for a successful end product. The first, and by far the most important, project development phase is that of design. Properly designed report writing programs not only provide the desired results but are easily constructed, implemented and maintained. The second phase of report writing is typically the construction phase and should be the easiest phase of the project if the design was properly attended to. Overall system considerations come into play here as well as the mechanics of the SAS data step. Finally, the implementation phase includes considerations such as who will be invoking the report writer, how, and with what data source. With implementation comes the need for macro based reporting and other sophisticated front-end control. Again, design is most important for a successful implementation.

Just a word about the level at which I have chosen to present this topic. I feel that it is most important to gain a conceptual understanding of the report writing process. I do not intend to teach syntax; the SAS Institute's documentation does a better job than I ever will. Although not wanting to over simplify a potentially complex process, I will clearly demonstrate the overall approach and how the SAS data step can be used as a relatively simply tool to accomplish effective report writing. Since data step report writing is simply the placement of data set information in a useful and self-explanatory form, the mechanics of what needs to be done are known to even the novice SAS programmer.

DESIGN PHASE

The design phase is by far the most important phase of any project. Report writing is no different. How many times have we had to rework or revise a programming project because the requester or the programmer simply did not spend the necessary time to determine what was actually needed. Experience has, or will have, taught every programmer this valuable lesson.

A SAS programmer should not entertain a request for a report generator without first receiving stringent specifications for the report. Spend time with the requester to detail what you expect from him or her prior to beginning the project. Where is the information to be reported on coming from? What form is it in: ASCII, EBCDIC, VSAM, Variable Blocked, Fixed Format, SAS data library? How do the data elements in the file relate to each other? What information will have to be derived and how? Many times the requester will not know the technical details involved in accomplishing the request. Send him or go with him to the people who do know. This information is vital to your design process. When the requester is the programmer these guidelines apply.

The requester should provide specifications on how the finished product may look. If possible, a mockup example
The title, headers, footers, and notes should be provided in as much detail as possible. What will the titles, headers, and footers look like and where on the page will they be located? This is not to imply that the requester need be intimately acquainted with the data, but simply have a clear idea as to what is desired regardless of any external problems. Initial specifications will give the programmer a better than average chance to produce an accurate and timely product.

With the conceptual design from the requester in place it is then time for the programmer to begin the logical design. A good logical design will significantly reduce the time it will take to code and modify the program. Always obtain a record layout and field definition of the raw data. If the report is to be based on an existing SAS data library obtain a PROC CONTENTS listing. By matching the data elements available with the requirements for the report variable, constant, and derived information can be readily identified.

Begin to plot the physical appearance of the report. On the page layout what is fixed and what is variable information? Of the variable information, identify which is available from the data and which will have to be input at invocation. Knowing this important information up front will have a major impact on how the code is constructed and implemented. Variable information may come from input data, data step manipulation, or innovative use of the macro facility at run-time. By designing the physical appearance of the report it will become obvious which SAS system options or data set options will be needed.

Titles, headers, footers, and notes are particularly important when considering their source and variability. These tend to be the most important part of the report. Above all, reports should be self-explanatory. If the reader cannot immediately discern what the report is all about then it will not be read. The TITLE statement provides little flexibility and does not allow the level of precision as does user defined formatting. If the TITLE statement is used one must match the line size system option (LS=value) to the logical report width so that centering is appropriate. Titting information may reside, and therefore be maintained, in a separate system file or as separate records in the data file of interest. Headers may be implicitly linked with the HEADER=LABEL option of the FITE statement or explicitly linked via the LINK or GOTO statements. The choice of which method is most appropriate for the reporting needs will dictate which of the several SAS techniques will be used. See Levine (1987) for a review of the report design. With a detailed sample report to review it should be readily obvious if all necessary requirements have been addressed. Is the report's appearance correct with respect to titling, page breaks, and calculated values? Initial sign off on the part of the requester is now appropriate.

**CONSTRUCTION PHASE**

Up to this point the programmer has not yet begun to enter SAS code into the machine. With a solid logical design in place, the actual coding can now begin. If the design included a formal flow chart then all the better, but many times an experienced SAS programmer can work with a detailed example of the required report and a good understanding of the data. This understanding should have been obtained during the design phase from a PROC CONTENTS listing of an existing library or a data dictionary of the raw data.

Prior to getting into the body of the report generator, the data step, the system environment impacts should be considered. For the most part these will be addressed with the OPTIONS statement to set any necessary options at the start of the program. The NODATE and NONUMBER options are often suggested since the data step will take control of these report features. In the simple case page numbering of the current
date is not required for the report and the default SAS System information should be prevented from appearing. Likewise, page length and page width need to be addressed with the PS= and LS= options respectively. If boldface type will be required in the report the OVP option, for overprinting, needs to be coded. The NOCAPS option will prevent lowercase input data from being translated to uppercase. Other system options are available to aid in explicitly defining the environment in which the report will be produced. It is highly recommended that required option settings be made explicitly in the event that the installation representative chooses to reset default values that are critical to the report.

For the most part raw data will be in a consistent and fixed format that is relatively easy to input with the SAS System. However, if this is not the case then consideration should be given to a front-end processor that will help to reduce the data to a more palatable form for the report writing program. Although the SAS System can read all forms of data, modularizing the data input process will significantly reduce the maintenance work which will inevitably be required. System sort routines, database extraction routines, or a separate SAS macro may be viable alternatives for this task.

In data step report writing there are really two issues that should be separated in the mind of the programmer: statements that manipulate the data for reporting and statements that actually do the report production. If the data is to be retrieved from an existing data library then the bulk of the first task is done. However, if raw data is being read in and reduced for the purpose of subsequent reporting than perhaps two or more data steps should be considered for the sake of simplified maintenance. Again, the concept of modularizing the data input process has relevancy to immediate and longer term efficiencies. Since the actual reporting data step need only work with a NULL data set, using valuable work space to contain multiple large SAS data sets is of no concern.

When creating or passing information for the purpose of reporting be aware of data set options and informational statements that may be needed to initialize the local environment. These statements should reside at the beginning of the data step. Data set options such as KEEP=, RENAME=, IN=, FIRSTOBS=, and OBS= are very helpful for defining the structure of a data set. SET/MERGE/UPDATE statement options that should not be overlooked are END= and NOBS=. When needing BY group processing in a report remember to initialize the FIRST.variable and LAST.variable capability with the BY statement. RETAIN, FORMAT, and LENGTH statements are good examples of informational statements that will prove indispensable.

When manipulating data for subsequent reporting do not ignore the possibility of SAS procedures as legitimate data reduction techniques. Efficiencies can be gained by using the output data sets from the FREQ, SUMMARY, or TRANSPOSE procedures, to name just a few. Many times the bulk of data step programming can be reduced to the simple merging of these output sets.

Another feature not utilized for data manipulation are the wide variety of SAS functions. When relying on data set variable values for conditional control of the report writer functions such as INDEX, LENGTH, VERIFY, SYMPUT, and SCAN are found to be indispensable. Other functions are available for format control: MONTH, YEAR, TODAY, PUT, TRIM, LEFT, RIGHT, and REPEAT.

Let us now turn to the actual report writing features of the data step, whether a stand-alone step or one that has been integrated with data manipulation. The key features of this process are the FILE and PUT statements.

The FILE statement is one of declaration in which the output destination is defined. The appearance of the final product is controllable with options used in this statement. The PRINT option defines the standard SAS print file as the destination rather than the global file reference. The LS= and PS= options perform the same function for the data step output environment as previously discussed for the global environment. NOTITLES suppresses the printing of default title lines. The LINESLEFT= option defines a single step variable which tracks the number of lines remaining on the current logical page. This is extremely useful when page break control is required. The HEADER= option defines a labelled section of code that is executed whenever SAS begins a new output page. This is a standard way of handling titles and column headings. The N= option defines the number of output lines available to the pointer. A special value of this option, N=PAGESIZE, instructs the end of output page available until implicitly or explicitly released. For multi-column reports this feature is very handy. Be aware that using the HEADER= and N=PS options together may cause problems if the page location for the header and PUT statements overlap.

Levin (1987) considers this problem as well as providing discussion on various ways of generating headings. The COLUMN= and LINE= options define variables that track the current pointer location. These are extremely useful for conditional pointer control using data step variable information.

The PUT statement is another cornerstone of data step report writing. With this statement both variable and literal constant information is selectively placed on the defined output file. Conceptually, I like to think of this statement as the antithesis of the INPUT statement with all, and more, of the features of Input. Pointer control is absolute or relative, both within a given line or across lines. This control may
be had with constant or variable values. If variable values are used, they may come from data values or other data step information. Although PUT statements typically execute once for each pass of the data step, they may be effectively incorporated in looping routines (DO, DO WHILE, or DO UNTIL) or conditional logic (IF/THEN/GO TO or IF/THEN/LINK). The possibilities for PUT logic are endless.

IMPLEMENTATION PHASE

Although presented as such here, the implementation of a periodic report generator is not in and of itself a stand-alone project phase as are the design and construction phases. Much forethought must be given to how a report generator will be invoked and maintained by and whom. At the point of conceptual design consideration must be given by the requester as to how the product is expected to be used. During the construction of source code the programmer must incorporate many of the features required of a successful implementation. These features may include modularity, substitution parameters, and complex conditional logic.

The operating system environment in which the program will be run is of primary consideration. Will it be driven by batch control language, interactive control, or a combination of both? Depending upon the periodicity of the runs, will it be executed through a production control process, spawned by another job, or generated ad lib by the user? The aim is to make the invocation of a periodic report writer as simple and painless as possible, but at the same time incorporating flexible function.

As an example let us consider a batch job where the job parameters are passed to the control language via an interactive session by the user. Immediate batch execution may be the result of such a session or execution held based on some predetermined schedule. The interactive session may be driven by most any language, e.g., TSO CLIST, CMS EXEC, or even SAS/AF. The important feature, whatever method, is that the user need not be conversant with any programming language, but merely know the requirements of the reporting request. In terms of the front-end processor any number of prompts may be required depending on the complexity of the request. Data files used, number of copies, or form type are only a sampling of the user controls possible in generating requisite job control language. Parameters required by the executable SAS code should also be provided at this time. These can be easily incorporated into the system input stream by a variety of techniques depending upon the operating system. The result should be that a given SAS report writer easily receives all necessary information for a successful execution.

Beyond the system control language necessary to invoke the report writer let us consider the use of the SAS Macro Facility to drive a report generator. At its most complex, the program will be a system of macros constructed as separate modules. Each module will serve a unique function within the process and will be driven by keyword or positional parameters. These parameters should have default values hardcoded, but may receive alternate values called by other macros. In many cases the macro calls will be conditional upon information passed at invocation, input data values, or conditions incurred during execution. Again, functional requirements of both the reporting program and how it is to be used will determine the level of complexity required.

Modular design of a macro system is recommended for ease of maintenance and simplified invocation. Use of the Autocall Facility will serve to streamline the entire process. This entails the creation of a separate library containing the macros necessary to run a reporting system. When allocating this library to the special fileref, SASAUTOS, in the control language all reporting functions are readily available to the program. If possible this library should be write protected for update by the system maintenance personnel only.

When using macros for report generation several items of note are worth keeping in mind, although they will not be considered in depth here. Macro processor activity establishes referencing environments; both a global and local environment exist within a SAS job containing macros. The definition of the environment for a given macro variable determines where it is available for use. When report writers need to transfer information contained in macro variables this becomes an important issue, particularly when using the SYMPUT and SYMPGET functions. Computer performance issues also need to be considered when using the macro facility for large and complex report writing. The use of macro variables for text substitution is more efficient than executing a macro to perform the same task. This may be the case for incorporating variable and conditional information in a report’s title. Macros are often misused in this regard.

More on these topics may be found in the SAS User’s Guide: Basics.

Automatic macro variables may also be utilized in the report proper or as a logging tool for the tracking of when and where a given report was produced. SYSDATE, SYSDAY, and SYSTIME provide temporal information related to actual execution and may be used in titles, footers, or logs of periodic reporting activities. SYSSCP and SYSVER indicate the operating systems and SAS System version under which the program executed. SYSVER indicates the character string defined by the SYSPARM= option and provides a useful method for passing information from the job control language to the executable code. This string could have been prompted for by the front-end processor and passed to the batch control
Any system of reporting macros should contain a "toolbox" of utility macros. Error checking routines should be included which will produce appropriate messages upon termination of a poorly executing report generator. The ERRORABEND option, ABORT statement, or the STOP statement should be considered for terminating execution when necessary. Not only should the invocation parameters be checked for appropriateness, but bad input data should not be allowed to produce an error ridden report. The only thing worse than not receiving the expected report is not knowing why. Another interesting utility macro is one which will produce a useful separator page to aid the distribution of multiple copies of the report. Unique utility macros in broader detail.

A few final notes on the implementation phase. First, although testing and debugging is an ongoing process in both the construction and implementation phase, a final test in conjunction with the user/requester is of vital importance. Both the user of the system and the recipient of the report need to understand possible pitfalls and future problems. Testing must be done regarding both human and machine resource efficiencies. This may require the development of several alternative prompting and reporting methods for comparing their relative merits. Second, prior to releasing the system consider the destination of both the SAS log and the system activity log. The use of several SAS options can limit or negate the production of a SAS log. These include, but are not limited to, NOSTREAM, NOCT, and NOSTACK. At minimum the output location for the log should be different from the actual report. Recipients of periodic production reports generally have no interest in seeing the SAS log. Perhaps the log should be written to disk with a finite retention period where it may be perused by responsible people in case of problems. Finally, documentation should be coalesced into a useable form for both the system maintenance person and the enduser. Not only should the user be instructed on how to invoke the needed report, but also provided with as much detail of the report generation process as is necessary for a good understanding of how the report is being produced. All too often the enduser is not given enough credit for needing to understand the workings of "their" program. A knowledgeable user becomes a valuable asset to the program maintenance function.

Throughout this presentation I have conceptualized the general process and tools needed by the SAS programmer to effectively produce reports. Although not limited to this report type, they are most important to consider when writing large production systems. The three phases of the report writing project as I have defined them, design, construction, and implementation, are important to consider as they relate to each other. The design phase begins with the receipt of a set of stringent specifications from the report requester. It is imperative that the individuals in need of a report be intimately involved in the design of the system that produces it. Their conceptual design should include a sample of the desired report so that the programmer has a place to begin when building a logical design. When the programmer can picture the end product on paper then the mechanics of the application are much clearer. Choice of methods for constructing the heading information as well as the body of the report becomes easy to make. The design phase should optimize the system for future needs. Finally, programming should not begin until both the programmer and the requester are clear about where the project is headed. What is done when coding begins must needs have already been identified and the required features of the SAS System are readily chosen.

System wide impact is the first to be addressed via the available SAS system options. These should be explicitly coded so that nothing is left to chance. The global environment must be adequately defined. By clearly segmenting functions within the program code future maintenance is simplified. The use of multiple data steps should not be overlooked as a way to meet this goal. Data manipulation requirements are many times distinctly different from those of the actual reporting code.

Many SAS statements that we are already accustomed to using are in the tool bag of data step reporting. Informational statements to initiate the step environment, conditional logic control statements, functions for data manipulation, and procedures for data reduction have all been used by the most novice of SAS programmers. Consider them to meet the needs of a report generator. The FILE and PUT statements are the cornerstones of data step report writing, but should provide no mystery for those that have used the INFILE and INPUT statements. Used in conjunction with looping and conditional logic routines these become very effective report writing tools.

Implementation of a report generator becomes an issue of who is going to use it and how. This phase can vary from the very simple, in the case of the programmer executing his own routines, to the very complex case of novice users requesting periodic, parameter driven reports. Operating system and SAS System environmental concerns...
need to be addressed. The SAS Macro Facility has been discussed as a very important tool to address the many implementation requirements through modular design. Again, maintenance headaches will be avoided as well as the invocation simplified. Two important issues relative to a happy report recipient are good quality error checking features and documentation. The programmer should not downplay the abilities of non-programmers. The client should be the most important partner in a venture of this type.

Data step report writing is not the ominous task novice programmers first see it to be. With a good conceptual understanding of what is involved and some basic SAS programming skills small projects will soon lead to the large and complex with no hesitation. My experience has shown that following the first successful project by the novice, there is no end to the enhancements and improved techniques to be tried by that programmer.

REFERENCES

Levine, Howard; Writing Customized Reports Using the Data Step; SUGI Proceedings 1987

Septoff, David and Hubbel, Katie; Developing and Using Macro Utility Libraries; SUGI Proceedings, 1986

The author may be contacted at
Blue Cross and Blue Shield of New Hampshire
2 Pillsbury Street, Concord, NH 03306

SAS and SAS/AF are registered trademarks of the SAS Institute Inc., Cary, NC, USA