Paper IN03_05

A Tour of the SAS Reporting Toolbox

Frank DiIorio, CodeCrafters Inc., Chapel Hill NC

Abstract

One of the most intriguing aspects of SAS programming is the variety of solutions that it allows for even the simplest tasks. Ask "n" programmers to add a column of numbers, and you'll get at least "n" solutions that are as valid as they are different.

Nowhere is this celebration of diversity more dramatic than communicating results, a/k/a "reporting." Be it a simple listing, statistical analysis, or graphic display of an analysis, the choice of tools available can be overwhelming to even seasoned SAS programmers.

This paper gives an overview of the more common reporting tools in Base SAS and SAS/GRAPH. We ignore products such as SAS/ASSIST and SAS/INSIGHT. These and similar inherently interactive tools ultimately have functionality limitations when applied to highly-formatted Real World reports requiring extensive data manipulation. It's best to just jump in and get your hands dirty with the DATA step *et al.*, because that's where you're likely to end up in the long run.

Rather than simply enumerate options and procedures, we focus first on a set of questions the programmer should ask prior to writing. Thus armed with a bit of focus and specificity, we examine the capabilities of the tools. We identify strengths and weaknesses of each, provide some examples and show how the tool "plays well with others."

This overview is just that – a high-level review of things to consider before programming and a look at what's available once coding starts. Readers should gain an appreciation of the scope of tools and their capabilities, and will hopefully consider trying different approaches to even old and familiar tasks.

Before You Write Code

Sometimes you get lucky. A SAS-literate client may, for example, ask you to "do a PROC PRINT of the adverse event data for clinic number 0122." That's simple – a LIBNAME, a PROC statement and a title, and you're done.

That's also boring. The only challenge is how fast you can write the program and verify that the results are correct. This scenario is also pretty unlikely, because "reporting" typically involves much more than writing the code for a reporting procedure.

This section identifies two broadly defined aspects of the reporting process that should be considered prior to writing the program. First we identify questions that should be asked as part of the **requirements-gathering** process. Then we describe some aspects of the **design and work flow** surrounding the program. The caveat that applies to the paper in general should be reiterated here – this is simply an overview of the requirements and process that you would typically use. It is *not* definitive (indeed, it's hard to envision a document that *could* be, given the breadth of the topic).

Requirements

Before you begin to write the report-writing program, you should know what the report will look like. This simple practice can be as overlooked as the need for it is obvious. Remarkably, and perhaps because programmers sometimes don't know what questions to ask, the requirements "cart" is often ahead of the coding "horse." Let's identify some features of the report that will have a bearing on the selection of tools that we'll be reviewing in the second half of the paper.

Detail and Order

One of the key features of any report is the level of **detail** that it contains (sometimes referred to as its "granularity"). Will it contain individual observations from the source data (e.g., employees)?

	race	sex	hireDate	salary	pctChange03
employee 1	x	x	mm/dd/yy	xxx,xxx	xxx.x%
employee 2	x	x	mm/dd/yy	xxx,xxx	xxx.x%
…					
employee N	x	x	mm/dd/yy	xxx,xxx	xxx.x%

Will it be some form of aggregation (e.g., counts of employees by race and department)?

	Avg. Salary	# of employees
race = 'White'		
department 1	xxx,xxx	xxx
department 2	xxx,xxx	xxx
…		
department N	xxx,xxx	xxx
race = 'non-White'		
department 1	xxx,xxx	xxx
department 2	xxx,xxx	xxx
…		
department N	xxx,xxx	xxx

Will the report contain a combination of individual and aggregate data? The combining scenario can be simple or complex. Individual values could be listed, preceded and / or followed by an aggregate measure. For example, employees could be listed by department, with subtotals for each department.

	race	sex	hireDate	salary	pctChange03
Department 1					
employee 1	x	x	mm/dd/yy	xxx,xxx	xxx.x%
employee 2	x	x	mm/dd/yy	xxx,xxx	xxx.x%
…					
employee N	x	x	mm/dd/yy	xxx,xxx	xxx.x%
Department summary:				*average*	*average change*
Department 2					
employee 1	x	x	mm/dd/yy	xxx,xxx	xxx.x%
employee 2	x	x	mm/dd/yy	xxx,xxx	xxx.x%
…					
employee N	x	x	mm/dd/yy	xxx,xxx	xxx.x%
Department summary:				*average*	*average change*

A more complex report layout may place summary data (the "executive summary" or "roll-up") first, followed by detail data. Reports using this format often insert navigational aids to assist the computer-based reader's movement through the document. These aids usually take the form of hyperlinks in the report proper or in a separate panel of the electronic document (represented below by dashed underlines):

	Salary	Age
Department 1	*dept 1 total*	*dept 1 avg*
Department 2	*dept 2 total*	*dept 2 avg*
Department 1		
employee 1	xxx,xxx	xx
employee 2	xxx,xxx	xx
…		
employee N	xxx,xxx	xx
Department 2		
employee 1	xxx,xxx	xx
employee 2	xxx,xxx	xx
…		
employee N	xxx,xxx	xx

The **order** of the data in the report also has a bearing on the programming strategy. Sometimes an obvious sequencing of individual observations presents itself. If we are listing data for individual employees or study subjects, it makes sense to order the data by last name or study identifier. For any sequencing field, consider the possible disconnect between the way it is *stored* in the data and the way it is *displayed* in the report. This often requires an extra step to maintain correct ordering.

Consider a data set where numeric variable GENDER represents males = 1 and females = 2. If the report specification required females to be listed before males (2's before 1's) and for values to be labeled as "Female" and "Male", we would need to create a separate variable for sort order or attach a user-written format to the data set. Here is one approach, using PROC REPORT as the display tool:

```
data report;
     set demog;
     if         gender = 2 then do;  genderSeq = 1;  genderText = 'Female';  end;
        else if gender = 1 then do;  genderSeq = 1;  genderText = 'Male';    end;
run;

proc report data=report;
     columns genderSeq lName genderText other vars go here;
     define genderSeq / order noprint;
     define lName     / order;
run;
```

Tools

Knowing report detail and order is vital. So, too, is being aware of programming resources that are at your disposal. This is distinct from the procedures and options discussed in the second half of this paper. The tools listed here are those that are typically available in many companies' programming environments. These are tools that routinize tasks and thus improve program development time and quality.

Code Library. Unless the report is unique and its rendering style previously untried, it's likely that there is a model program upon which to build. Phrased another way, if your program name is TABLE18.SAS, there's a good chance that you can reuse some of the earlier programs or at least get some idea of how the other 17 tables were handled. This is an important point, because it not only speeds reliable development, it also makes subsequent modifications and debugging simpler – one development approach throughout a project means, in turn, one modification approach.

Macros. Extant code or not, you should also look for generalized tools to perform portions of the report programming. Consider a typical requirement of a pharmaceutical application's table programming: for distinct levels of a variable, count the number of patients, distinguishing treatment types. Also include each category's percent of total for the treatment group. It's not difficult to write code from scratch that will provide the numbers, but it is such a basic task that you may already have a macro that will handle this. The importance of these tools cannot be overstated, for they provide a reliable, validated means for quickly producing results.

ODS. Some of the most demanding features of the report can be purely presentation-related. Fonts need to be adjusted, column headers need a different color background, and so on. These display issues can often be handled by the SAS Output Delivery System (ODS). You can create new styles and table layouts or modify those that are already part of SAS software. If the report in question is recurring or part of a series of related reports, however, it's likely that a style or table definition already exists. Here, as with macros, it pays to see what has already been developed, unless, of course, you want the distinction of independently reinventing the style "wheel."

Rendering

Programmers of a Certain Age who cut their report-writing teeth on plain text files had comfort in the knowledge that a page was usually 132 print characters across by 60 lines down. This made any specialized placement of text on a page relatively easy. The resulting text was, to put it mildly, usually boring to look at. Only with a great deal of programming effort could rendering features such as boldfacing or italicizing be inserted. Color was not a possibility, nor was non-linear reading of the output.

The possibilities for rendering a display are so numerous today that it often seems coding is more focused on data *display* than data *content* (preoccupation with "sizzle" rather than "steak"). Writing to one or more of RTF, PDF, or HTML has significant implications for programming approach. Even with the ODS shouldering much of the display burden, each style has its own considerations and trade-offs.

Even if the formatting style of the report is predetermined, keep in mind a few caveats. First, PROC output will vary by the formatting you select. This at first glance seems self-evident. Output written as a web page *should* look different from that written to a PDF. This is, in fact, what happens, and the differences are predictable and documented. What is less obvious, and what can lead to intense amounts of workaround coding, are rendering issues that may more rightly be designated as a bug or design flaw.

The REPORT procedure has a good example of this. If a group is continued onto a second page, the current group value is displayed on the first line of the second through last continuation pages. If the same program is used for writing to an RTF or PDF file, the continuation text is *not* written. Thus if the report specification requires RTF or PDF and asks for continuation text, it's time to write what is often an inexplicably large amount of code to mimic what was the default in the simpler and plainer text file.

Extra Coding. Bugs aside, another point to remember when chosing rendering styles is the amount of extra programming that would be required to accommodate the format. An issue dovetailing from this is whether the report needs to be rendered in multiple formats. If so, this may require special handling of one or more variables.

Consider the case of a report being written to both plain text and HTML. The plain text file can simply display a variable containing a file name as-is. If the report specification asks for the web page to contain a hyperlink to the file, HTML tags must be added to the variable. This imposes an extra requirement on the programmer to be conversant with basic HTML syntax. It also requires that a new variable be created and that *it* and *not* the original file name be used in the REPORT code that writes the HTML. Thus we see that a conceptually simple request becomes perhaps a tad trickier than expected.

Different Media. Finally, let's note a design construct that cuts across all rendering styles: consideration of both printed and on-screen output. If output is written specifically for one medium, then the display can be fine-tuned for it. If, however, the report is likely to be viewed on screen and then printed, the design must consider the more constrained and less capable medium – that is, the printer.

Many of the features that make screen-based displays appealing are lost in the translation to paper: color printed to anything but a color laser becomes gray-scaled; hyperlinks and animated GIFs become "non-hyper" and "unaminated"; text that scrolls down or across multiple screens is interrupted, possibly without column headings, on paper. Ensure that both the screen and hard-copy versions of the report are acceptable, and consider the possibility that two separate reports may be needed.

Control and Specifics

Report specifications can be wonderfully vague or amazingly detailed about matters of display. Regardless of whether you have specs or not, eventually you will have to make decisions that are answered or bypassed by the spec. The sooner in the report-writing process that you are aware of some of the possible issues, the better. Here are some items that were not covered in the previous sections:

- **Within-column formatting.** The amount of precision for statistical displays usually depends on the type of statistic. If different types of statistics are in the same column, then some preprocessing will likely be necessary to control the number of decimal places being displayed, and to align values on the different rows' decimal points.
- **Page and font specifics.** The sooner the font family and point size of the display is determined, the better. These and other features such as page orientation, margins, and standard headings should be agreed upon early in the programming process, if only to identify abstract report layouts that may be impossible to implement. For example, a statistician may want N's and percentages for each of 5 treatment groups, but the combination of print positions for row headers and five groups of N's and percents may not fit, once the positions available after margins and fonts are considered. Remember: redesign is easier and faster in the program development process.
- **Missing values.** Ask how missing values should be represented. Will there be special handling for special missings .a through .z? These are often easily handled by the MISSING system option, but bear in mind that the option accepts only a single character. If the requirement is to fill the entire display column with hyphens, for instance, the programming task becomes more complex. If a variable contains multiple fields, say N and percent, and the N is missing, should anything be printed for the percent? As you examine the specification, remember that no question is too minute, and that questions asked early in the programming life cycle pay greater dividends than those asked at 10:00 pm the day the report is due.
- **Incomplete categories ("missing observations").** Just as we can have incomplete data for a row, be it an individual observation or an aggregate, we can also have what are, in effect, missing observations. If possible responses for an event are "Mild" "Moderate" "Severe" and "Fatal" and actual responses captured in the data do not include the latter two categories, should the table contain them? There is no single correct answer, and so it's always good practice to pose the question. (Note that the answer may be contained not in a project-specific document but in a corporate-level SOP.) Missing observations require special handling by the programmer: the missing categories must be inserted into the analysis data used for reporting. Likewise, values in the analysis columns must correctly identify missing values.

Design and Process

Division of Labor

One of the great temptations for a novice SAS programmer is to examine the lengthy list of reporting tools and procedures and conclude that these are the entirety of the reporting tool set. As we will see in the second half of the paper, the reporting tools are usually the end point of a series of steps. For example, a DATA step may add classification variables to a data set, then the SORT procedure rearranges the data by one of the new variables, finally followed by the REPORT procedure. While it might be possible to do everything within the confines of REPORT, it is more likely, and usually more efficient, to make multiple passes through the data, ultimately handing REPORT a data set that requires little special processing.

The word "efficient" in the previous sentence is a loaded one. It could be argued that if REPORT (or any procedure) could perform the intermediate steps of grouping, sorting, and so on, then it should be allowed to. This is, after all, what the procedure was designed for. Over

time, however, many programmers tend to adopt strategies that divide the work between the reporting procedure and other tools. Arguments in favor of the divide and conquer approach are:

- **Debugging.** Most reporting procedures lack debugging features. Other than having a clear understanding of the documentation and your data, using the tool is largely a matter of trust that the "black box" performs as expected. Consider how you would react if the tool didn't work as expected. If, for example, the sort order wasn't as expected and you were missing groups, you would likely insert a debugging DATA step or CONTENTS or PRINT procedures to investigate. Over time, many programmers build sorting and grouping variables themselves, rather than letting the reporting procedure make these decisions. This lets the variables be examined independently of the reporting tool, and lessens dependence on a single reporting technique. The multiple step approach also figures in the next two items in our list.
- **Control.** Programmers are by nature interested in control of a process. By moving select pieces of the reporting process out of the reporting tool, the programmer regains control over how a variable, and in turn, a report, will appear. This is not a purely psychological advantage: if the report specification's demands outstrip the ability of the reporting tool, then preprocessing is required. The workaround for PROC REPORT's inability to repeat group variable values at the start of a PDF continuation page (noted earlier) is a good example.
- **Documentation.** When group, break, and other variables are calculated manually, their exact meaning is known. Similarly, if any data set reshaping or reordering takes place using the SORT or TRANSPOSE procedures, the precise sequence of steps (and attendant diagnostic code) is clearly spelled out. The reporting process is, in turn, supported with additional internal documentation.
- **Efficiency.** Efficiency is at least a two-pronged concept, encompassing programmer and computer resources. From a *programming* perspective, writing multiple steps takes more time and usually results in more "moving parts" than trying to jam everything into a procedure and maybe a DATA step. The start-up cost may be high due to any preprocessing code. The strategy pays off in the long run, since program debugging and enhancements can be readily slotted in to the program framework.
  From the *computer resource* perspective, the extra steps likely don't consume appreciably more time, and may even execute faster than a single procedure. This unlikely scenario can come to pass when a reporting procedure (notably TABULATE and REPORT) is called upon to perform several tasks that it can legitimately perform, such as sorting or calculating statistics, but could be done more efficiently by other procedures (SORT, MEANS, etc.). In these cases, it would be faster to sort or calculate using other procedures, then pass the results to the reporting procedure. In short, play to the strengths of the individual procedures, even if it means multiple program steps.

There are, of course, no absolutes. In the workaday world, the typical program balances reporting tool-specific features and preprocessing DATA steps and procs. The balance is a matter of instinct and thus cannot be taught *per se*. Simply being aware of the tradeoffs described here is a good start toward developing the instinct.

Sequence

The previous section described what might be loosely called a design strategy for all but the simplest reporting tasks. Our attention now turns to just how the strategy should be implemented. It's not as straightforward as it appears. Even if we preprocess with DATA steps and procedures before running the reporting tool, there are some subtleties at play.

The most important aspect of the actual coding process is to write the program in discrete, testable pieces. Assume for a minute that the *opposite* strategy is in effect: you correctly see the need to calculate a grouping variable, then transpose the data by drug treatment group,

assign formats to control labeling, set up the statements necessary for PDF output, and run the TABULATE procedure.

You write the program, run it, and you get … gibberish. Quickly, now: what went wrong? Was it the grouping variable? Did we transpose correctly? Was the format definition sufficiently inclusive, or were there "orphaned" groups? You may get lucky and see the error right away, but it's more likely that you won't.

Rather than jump in and write the entire program at once, think of program-building as you would the construction of a house. With a blueprint in hand (program spec, plus answered questions regarding missing values, formatting, etc.), build the foundation. Convince yourself that the input data to the report has the values, range of classifiers, and other attributes that you expect. (This is why the divide-and-conquer approach to reporting is appealing: you can create a temporary data set, then selectively apply PRINT, FREQ, and other procedures as needed to get the comfort level you need with the data.)

When the data is in the correct format, you can "frame in" the report: first get the principal features correct. Are the rows in the correct order? Are missing categories accounted for? These and similar features are key, for if they are suspect, the remainder of the report, no matter how nicely formatted, is pretty much worthless. Finally, the programmatic equivalent of paint, carpet, and foundation plantings can be addressed: title fonts, decimal point alignment, and the like. And to complete the analogy, program validation is the equivalent of the building inspector, providing your certificate of occupancy.

Again, just as there were no absolutes in the previous section's suggestion for division of programming labor, we see here possibility for variation. If, for example, the report requires creation of hyperlinks in HTML, a good strategy would be to read the data, build the variable containing the link, and write a simple "proof of concept" report that lets you test the link. In general, the programming strategy should first address what you think are the report's most important and potentially problematic features.

Toolset Review

Let's turn our attention now from matters of design to the actual tools you need to craft a report. The first sections of this paper emphasized the need for careful examination of the components of the report. Since we emphasized the merits of multiple steps, the "divide and conquer" method, we'll discuss the reporting tools in roughly the chronological, development order that you use them.

For each tool, we'll discuss its use in report writing and look at brief examples. Since the audience for this paper is the programmer relatively new to SAS, the discussion avoids the "whiz bang" capabilities, sticking instead to more typical applications.

Recall that the purpose of this paper is to lay out the Big Picture of report writing. Thus the tools are discussed in general terms, as parts of a whole, rather than in syntax-level detail.

The Big Picture

If the exact content of the report-writing toolset is "somewhat" arguable, then the grouping of the tools is *really* arguable. Nevertheless, **Figure 1** (next page) maps one way to see the tools and how they might interact. Note the groupings (white print on black background), the arrows, and the box with the dashed lines. We see numerous relationships, among them:

- The **DATA step** and various **non-reporting PROCs** such as SORT and TRANSPOSE interact with each other and pass data to reporting PROCs. Using these tools prepares the data for display, and is at the heart of the divide-and-conquer strategy.
- **Graphic and text-based displays** can accept data from other, **statistically-oriented procedures**. The strengths of each group can be exploited with this approach: stat procedure output can be captured as data sets or ODS output objects, which can be used

as input to the graphic or text-based reporting procedure. By using multiple procedures just as we features from the previous bulleted point, we let each procedure do what it knows best – analyzing data or displaying it.

- **ODS** and **environmental settings** and statements affect the rendering and organization of report output.
- The **macro language** can influence everything on the map. Its ability to conditionally execute code, generate program statements dynamically, and influence the content and appearance of the report make it an indispensible tool for complicated reports or for generating groups of distinct reports with a common theme.

The DATA Step

Statements in the DATA step can be used to create variables to signal page breaks in reporting procedures, create subtotals, change data set granularity, etc. The DATA step can also write the report, using PUT statements, or create graphic annotations for SAS/GRAPH output. The DATA step's versatile syntax allows you to do pretty much anything imaginable (as well as make pretty much any *mistake* imaginable). It's an ideal tool for reports where complete control over monospaced fonts is required. Its appeal is heightened by the wealth of debugging tools and techniques, both of which are partly attributable to it being at the heart of SAS programming since its inception.

Examples

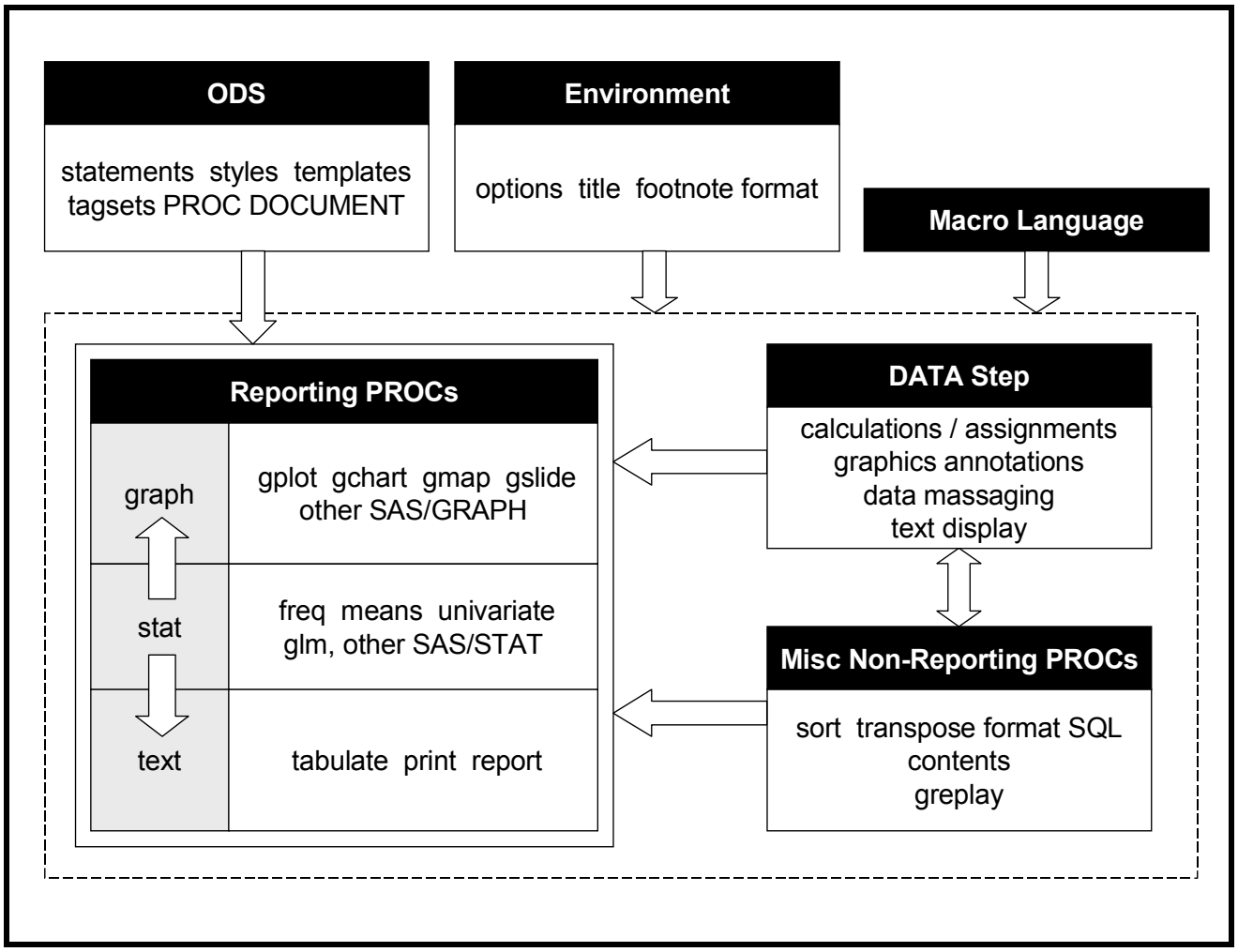
Read data set MASTER and write to file REPT1.TXT. Keep track of BY group variable CLINIC. At the start of a new value of CLINIC, go to a new page if there are fewer than 5 lines available on the current page, and reset the CLINCOUNT counter. Write values for each observation, and display subtotals and grand total at the end of a CLINIC group and at the end of the data set.

```
data _null_;
     set master end=eof;
     by clinic;
     file 'rept1.txt' linesleft=ll;
     if first.clinic & ll < 5 then put _page_;
     if first.clinic then do;
        clinCount = 0;
        put // 'Clinic' clinic;
        end;
     clinCount + 1;
     put report detail lines;
     if last.clinic then put "Subtotal: " clinCount comma5.;
     if eof then put / 'Total count: ' _n_ comma5.;
run;
```

Miscellaneous (Non-Report) PROCs

SORT: Provides exact control over the order of observations in a data set, optionally eliminating duplicates. **TRANSPOSE:** Changes data set rows to columns. This can be helpful when post-processing the results of a statistical procedure, readying the data for display. **FORMAT:** User-written formats provide a means for mapping one or more values of a variable to a new, "formatted" value (e.g., map a value of 1 to "Yes" and 0 to "No" or 1 through 100 to "Lower" and 101 through 200 to "Upper"). Formats are indispensible for creating new variables or for displaying stored values in a more "conversational" manner. **SQL:** PROC SQL is SAS's version of the industry-standard Structured Query Language. The procedure's syntax allows complex data manipulation utilizing SQL's compact notation. There are several SAS extensions to the language, notably the ability to "pass through" commands to non-SAS data sources and to store query results in macro variables. **CONTENTS:** Useful for identifying just what, exactly, you created. It lists data set and variable characteristics – the number of observations and variables, variable type, length, label, etc. **GREPLAY:** Displays the stored results of SAS/GRAPH procedures, playing them into a template and allowing multiple charts and graphs

Figure 1 Report-Writing Tools



to be displayed on a printed page or image.

Examples

Use SQL to write macro variables T0, T1, and T2. These contain the counts of treatment variable TRT categories 0, 1, and 2. Later, in the REPORT procedure, use these macro variables in the column headers for their respective column variables.

```
proc sql noprint;
     select sum(trt=0), sum(trt=1), sum(trt=2)
     into   :t0,        :t1,        :t2
     from demog;
quit;

proc report split="~" procedure options go here … ;
     columns rowHeader treat0-treat2;
     define treat0 / "Placebo~(N=&t0.)";
     define treat1 / "1 mg~(N=&t1.)";
     define treat2 / "5 mg~(N=&t2.)";
run;
```

Sort data set INVENTORY by SKU within LOC, then process it in a DATA step, identifying duplicate LOC-SKU combinations with a message in the SAS Log. Write only the first LOC-SKU observation to data set INVENTORYUNIQUE.

```
proc sort data=inventory;
```

```
     by loc sku;
run;

data inventoryUnique;
     set inventory;
     by loc sku;
     if ^(first.sku and last.sku) then do;
        put message about duplicate loc-sku combinations;
        end;
     if first.sku then output;
run;
```

Create numeric format TMT. It maps numeric values 0 through 3 to descriptive text strings. The format can be used to replace the numeric values with text when used in a reporting procedure. It can also define a variable's complete range of values and affect the contents of a PROC MEANS output data set. (See example in "Reporting PROCs (Statistical)," below, for an example using the format for this purpose.)

```
proc format;
     value tmt 0 = 'Placebo'  1 = '1 mg'  2 = '5 mg'  3 = '10 mg';
run;
```

Create a SAS/GRAPH display template. It contains four areas, which can be filled by any graphics procedure output (see "Reporting PROCs (Graphic)," below, for an example using the template).

```
proc greplay tc=reflib.tempcat nofs;
tdef newtempl
     1/llx=0  lly=  ulx=0  uly=50  urx=50  ury=50  lrx=50  lry=10
       color=blue
     2/llx=0  lly=50  ulx=0  uly=90  urx=50  ury=90  lrx=50  lry=50
       color=red
     3/llx=50  lly=50  ulx=50  uly=90  urx=100  ury=90  lrx=100 lry=50
       color=green
     4/llx=50  lly=10  ulx=50  uly=50  urx=100  ury=50  lrx=100 lry=10
       color=cyan
run;
```

Reporting PROCs (Statistical)

The procedures discussed here can be used as standalone reporting procedures, producing reasonable output by default and attractive output when rendered with ODS styles. We place emphasis here on their role in summarizing data. For example, MEANS could read a data set with transaction-level data and collapse it into one observation per region with variables holding means, counts, and averages for several variables.

FREQ, **MEANS**, and **UNIVARIATE** can create data sets with summary statistics. The output data set can be a single observation, representing the selected statistics across all of the input data set's observations. The output data set's observations can also represent levels of a classification variable or combinations of levels of two or more classifiers. Some of the possibilities are shown in the code fragments below.

Program Fragment	An observation in the output data set represents …
`proc freq data=inv;` `  tables sku / out=skuCount;`	A non-missing SKU number

`proc freq data=inv;`
`  by region;`
`  tables sku / out=skuCount;`
	The combination of REGION and SKU (percentages add up to 100% within each level of REGION)

`proc means data=inv;`
`  class region sku;`
`  var onHand;`
`  output out=counts sum=;`
	Four levels of summary, distinguished by variable _TYPE_: 0 = summary for the entire data set; 1 = summary for each non-missing value of SKU; 2 = summary for each non-missing value of REGION; 3 = summary for each non-missing combination of

Program Fragment	An observation in the output data set represents …
	REGION and SKU.

`proc means data=inv nway`
`  class region sku;`
`  var onHand;`
`  output out=counts sum=;`
	Same as above, but since NWAY was specified in the PROC statement, only for _TYPE_ = 3.

The layout of these and other procedures' output data sets is predictable and documented. Refer to the SAS online documentation or other resources for details.

These procs and GLM and other statistical procedures can also have output selectively captured as data sets by using the ODS SELECT statement. The content of these data sets, both variable and granularity, is predictable and well-documented. The data can be used in a variety of scenarios. It could be used by itself, using a reporting tool such as PROC REPORT to, in effect, "dress up" the values. It could also be combined with individual observations from the input data, then passed to a reporting procedure.

Examples

Read data set SALES and obtain the crosstab (two-way frequency) of REGION and TGROUP. Store the result in data set FREQS (we know from the online help that it contains the analysis variables, plus cell count variable COUNT and percent of total, variable PERCENT). Postprocess FREQS, creating character variable NPCT, which has the form NNNNN (X.XX%). We play to the tools' strengths: FREQ counts categories, the DATA step massages the data and makes it ready to use in a reporting procedure.

```
proc freq data=sales noprint;
     tables region * tGroup / out=freqs;
run;

data rept;
     set freqs;
     length nPct $14;
     nPct = put(count, 5.) || " (" || put(percent,4.2) || "%)":
run;
```

Summarize CLIN by treatment group variable TMT. Use the format created in "Miscellaneous (non-Report) PROCs," above, to ensure that output data set SUMMCLIN will contain every *expected* level of TMT, not just those actually *present* in the input data set. The output data set will contain the number of missing values of CLIN variables EVENT and SIGEVENT. Once SUMMCLIN is created, it can be input to a reporting procedure.

```
proc means data=clin noprint nway completeTypes;
     class tmt / preloadfmt;
     var event sigEvent;
     output out=summClin nMiss=missEvent missSig;
     format tmt tmt.;
run;
```

Summarize CLIN by all observed combinations of RACE and SEX. Output data set EXTREMES contains variables RACE, SEX, and the five minimum and five maximum ages for each RACE-SEX combination (variables YOUNGEST1 to YOUNGEST5 and OLDEST1 to OLDEST5). As in the example above, the output data set can be displayed using a reporting procedure or, of course, can be merged or combined with another data set.

```
proc means data=clin missing noprint nway;
     class race sex;
     output out=extremes
            idGroup(min(age) out[5](age)=youngest)
            idGroup(max(age) out[5](age)=oldest)
            ;
run;
```

Reporting PROCs (Text-Based)

These are the procedures that you immediately think of when you hear "reporting tools." By now, you can see that they are simply the end-point, rather than the entire scope, of the report-writing journey.

PRINT: This is the original reporting procedure, and still the simplest one to use. Although you can exert some control over the rendering of its output (see "ODS," below), the most common use of PRINT is for quick, straightforward listing of variables. The syntax provides control over the order of variables, their formatting, and text contained in column headers. PRINT also provides some rudimentary subtotaling capabilities. It's best to think of PRINT as a tool for simply listing observations (in contrast with other tools, described below).

In the context of the divide and conquer strategy, think of PRINT as a diagnostic tool used for displaying data sets, answering such questions as "Are my BY variables sorted correctly?" and (my favorite) "What in the world did TRANSPOSE do to my data?"

Examples

Simplicity itself. Display select variables in TEST, using variable labels as column headers.

```
proc print data=test label;
     var subjID file fileHREF;
run;
```

REPORT: Rather than over-burden PRINT with a host of report-writing features, SAS software designers wisely chose in the mid-1980's to force programmers to make a conscious decision for data display: do they want to have a simple tool to quickly list data (PROC PRINT), or do they need something that will allow significant control over detail lines and permit on-the-fly calculations and breakpoint calculations (PROC REPORT)?

The REPORT procedure has an interface that lets you build and save report components using menus, but it is more effectively used in the non-windowing environment (syntax referred to as the "batch language"). Like PRINT, REPORT recognizes ODS styles and templates. The similarity ends there. Among REPORT's capabilities are:

- Simple specification of multiple column reports (plain text only)
- Headers that span multiple columns
- Significant computational and text display capability at break points within and following the report
- Features for most ODS destinations that allows you to exercise significant control over how report rows or even individual cells are rendered.

These and other desirable features exact a small mental cost. The syntax and logic can be hard to grasp at first. REPORT also has features that are appropriate for one type of output but are ignored, behave differently, or cause an error in a different output environment (for example, the FLOW option is valid for plain text output, but is ignored when creating a PDF). The requirements for some reports, especially when dealing with page breaks and continuation text, are such that it's often easier to create new variables for these purposes than out-flank the restrictions imposed by the procedure. Even considering these issues, this is likely to be the Procedure of First Resort for all but the most demanding specifications.

Examples

Display three variables from data set TEMP using all default width, format, and column labeling options. Create a phonebook-like display, allowing up to five columns if space is available.

```
proc report data=temp panels=5 noWindows;
     columns subjID tmt nAE;
run;
```

Create a report from data set TEMP. The text "ID Fields" spans the columns for variables SUBJID, SEQNO, and TMT. Use DEFINE statements to supply text for column headers and to override formats stored with the data set.

```
proc report data=temp noWindows;
     columns ('ID Fields' subjID seqNo tmt) baseDT termDT termCode;
     define baseDT / format=date9. 'Baseline';
     define termDT / format=date9. 'Termination';
     define termCode / format=tCode.;
run;
```

Create a PDF similar to the previous example. This time, however, we take more control over the rendering, selecting Arial 8 point as the default for the data and reserving a column width of 2.5 inches for variable TERMCODE.

```
ods pdf file='status.pdf';
proc report data=temp noWindows
     style(column)=[Font_Face=Arial Font_Size=8pt]
     ;
     columns ('ID Fields' subjID seqNo tmt) baseDT termDT termCode;
     define baseDT / format=date9. 'Baseline';
     define termDT / format=date9. 'Termination';
     define termCode / format=tCode. style={cellWidth=2.5in};
run;
ods pdf close;
```

Take advantage of the display capabilities of the REPORT procedure. First, pass through PERM.ENROLL in a DATA step. Variable PROBLEM is set to 1 if there are any data issues for the observation, otherwise it is 0. Use the ODS to open ENROLLSTAT.PDF. The COMPUTE block instructs REPORT to set the background of a row (i.e., an observation) to light red if the sum of variable PROBLEM is 1. Thus any observations with problematic data will stand out when reviewing the report. Notable, too, is the choice of "light red," rather than simply "red." Since this report will be used in printed as well as viewable formats, we had to consider how the report appeared on paper. Red was too dark, but light red was readable.

```
data tempEnroll;
     set perm.enroll;
     create variable PROBLEM – 0=obs. is clean, 1=data problem
run;

ods pdf file='enrollStat.pdf';
proc report data=tempEnroll noWindows;
     var subjID tmt enrollDt baseDt termDt problem tempVar;
     define problem / noPrint;
     define tempVar / computed noPrint;
     compute tempVar;
        if problem.sum = 1 then
           call define(_row_, 'style', 'style=[background=lightRed]');
     endComp;
run;
```

TABULATE: Few procedures in SAS software are as polarizing as TABULATE. People tend to love it or avoid it at all costs. Much of both the enthusiasm and angst is attributable to its somewhat arcane syntax, but once it's mastered, you can produce complex data summaries with remarkably little coding effort. The procedure's strength is implied by its name – it is a tool for displaying summary statistics, optionally within or across multiple classification variables. It can also produce rows or columns in the table that automatically summarize across all displayed levels of a classification variable, and can calculate group and overall percentages. You can also exert considerable control over all or part of the table's appearance via ODS styles and user-written formats.

TABULATE should usually *not* be considered a tool for listing individual observations in a data set. That task is better left to PRINT and REPORT.

Examples

Calculate the sum of EXPENDITURES. Rows represent distinct combinations of DIVISION nested within REGION. Columns represent discrete values of TYPE. Thus a cell could represent, say, the sum of EXPENDITURE for TYPE=1 and REGION="East", DIVISION=2.

```
proc tabulate data=energy;
   class region division type;
   var expenditures;
   table region*division,      /* Row dimension */
         type*expenditures /   /* Column dimension */
         rts=20;
run;
```

TABULATE shines when it is computing percentages. In this example, rows are individual and overall TEAM values, and columns are the sum of SALES and various percentages. Cells represent the percent of total sales by TEAM "A" in CLASSROOM "1" within all TEAM "A" (ROWPCTSUM), across all CLASSROOM "1" (COLPCTSUM), and overall, across all teams and classrooms (REPPCTSUM). As we said earlier, the syntax is not obvious, but worth the struggle.

```
proc tabulate format=7.;
     class team classrm;
     var sales;
     table (team all),
     classrm='Classroom'*sales=' '*(sum
             colpctsum*f=pctfmt9.
             rowpctsum*f=pctfmt9.
             reppctsum*f=pctfmt9.)
             all*sales*sum=' '
             / rts=20;
run;
```

Reporting PROCs (Graphic)

SAS/GRAPH software makes it fairly easy to generate attractive displays that can be embedded in documents, hyperlinked to other pieces of SAS output, and the like. In some cases, Base SAS procedure calls can be used almost as-is to produce graphic displays. Write a GOPTIONS statement to describe the location and size of your JPG file, then write "PROG GPLOT" instead of "PROC PLOT", and you're good to go. GCHART and CHART also have syntactical similarities. Finally, GPRINT, rather than being the pixellated cousin of the PRINT procedure, will convert *any* plain text file into a graphic image.

The true power and programming challenges of the graphics procedures come into play when the procedures are used for reports that go beyond simple scatterplots and bar charts. A little additional preprocessing prior to procedure calls enables you to:

- Generate HTML with "hot spots" that link to other files
- Overlay explantory text and figures on a display
- Create animated GIFs
- Combine multiple displays in a single page using templates

Examples

Create a horizontal bar chart, one bar per value of REGION. Bar width is proportional to the sum of SALES. GCHART uses the PATTERN statement's specification of red fill for the bars. Notice that in this case there is *no* difference between the graphic-oriented GHCART's syntax and that of its character-based predecessor, CHART.

```
pattern1 color=red;
proc gchart data=sale2005;
   format sales dollar8.;
   hbar region / sumvar=sales;
run;
```

Combine four charts into a single display using the template defined in "Miscellaneous (non-Reporting) PROCs," above.

```
run GCHART, creating four charts (default names in the output
catalog are GCHART, GCHART1 … GCHART3)

proc greplay igout=grafcat gout=excat tc=tempcat nofs;
             template=newtempl;
treplay 1:gchart
        2:gchart1
        3:gchart2
        4:gchart3;
quit;
```

Environment

No matter how much control is available to you in the DATA step, utility procedures, and reporting tools, your programming task is often made simple (indeed, made *possible*) by your being familiar with SAS features that are beyond the confines of the DATA step and various procedures. These so-called "open code" statements can set header and footer information, limit the number of observations available for processing, and perform other chores that assist the development and debugging process. For all but a few of these features, bear in mind that they can be set and re-set multiple times within a program. In some cases, they can be in effect only for the duration of a procedure. Your realtor's "location, location, location" mantra is applicable here as well.

Here are some statements that influence the programming environment. The list is limited to items most frequently used by the novice SAS programmers in this paper's audience.

TITLE and FOOTNOTE. These statements allow you to specify up to 10 lines of header and footer information. When used with a PROC that is performing BY group processing, the special text BYVARn and BYVALn in a title statement inserts the variable name and value, respectively, of the nth variable in the BY statement list. This adds considerable descriptive value to the output header. All output destinations allow you to selectively left, center, and right-justify portions of the text to display. All destinations other than plain text also provide formatting options so that you can control the color and size of text.

Even such straightforward features as these have subtleties when writing to PDF files. The maximum number of title or footnote lines is 10 for any output destination. However, if a title or footnote contains the PDF line break characters (`n), the output will be broken into multiple lines. The need for more than 10 lines of headers or footers is infrequent, but simply knowing you *can* circumvent the limit, if only for PDFs, is one of those "nice to know it's there" nuggets.

Examples

Create titles with some rendering instructions. If the type of instruction is inappropriate for the output destination – H=2 doesn't make sense for plain text – it is ignored without a NOTE or WARNING in the SAS Log. The second title line inserts the current value of the BY variable SALESREGION. The footnote uses both user-created and system macro variables (PROG and SYSTIME, respectively).

```
proc report options omitted … ;
by salesRegion;
title1 j=l "CureAll Pharma, Inc."  j=r "Wonder Drug NDA";
title2 j=c h=2 "Table 1: Patient List for Region #byval1";
footnote1 "Program: &prog." j=r "Run Time: &sysdate. at &systime.";
```

OPTIONS. SAS system options control a host of features in the program environment. The SAS online help files contain their definitive enumeration. The following table lists some of the more useful options for report development:

Option	*Comments*

Option	*Comments*
FIRSTOBS OBS	Specifies the first and last observations to process from a data set. Can be used globally, in an OPTIONS statement, or can apply to a specific data set (as a data set option). Useful during development and debugging.
ORIENTATION TOPMARGIN BOTTOMMARGIN RIGHTMARGIN LEFTMARGIN BINDING	These options control page layout. They are relevant for print destinations that support the concept of a printed page (RTF, PDF). The HTML destination does not create pages, and will ignore these options without printing NOTEs or WARNINGs. Be careful to specify these options *before* using any ODS statements that open the print destination.
PAGESIZE LINESIZE	For plain text destinations, controls the number of horizontal and vertical print positions. If TOPMARGIN and BOTTOMMARGIN are specified after PAGESIZE, they will reset PAGESIZE to a value consistent with the margins and default font. Likewise, if LEFTMARGIN and RIGHTMARGIN specifications follow LINESIZE, they will reset it. The override occurs quietly – no NOTEs are written to the SAS Log. These options are ignored without NOTEs or WARNINGs for other types of output.
BYLINE	Controls whether BY-group information is displayed in output when a procedure is run with a BY statement. See "TITLE and FOOTNOTE," above, for a cleaner, more readable means of inserting name and values of BY variables into the output.
CENTER DATE NUMBER PAGENO	Control basic features common to all output types: horizontal centering of output; display of the date, display of the page number, and resetting of a displayed page number.

Examples

Use the OPTIONS statement to define the page setup. Then use OBS as a data set (not globally-applicable) option for testing.

```
options pageNo=1 noDate orientation=landscape
        topMargin="1.5in" bottomMargin="1.0in"
        leftMargin="1.0in" rightMargin="1.0in"
        ;

proc report data=testData(obs=100);
     report specifications go here
run;
```

LEGEND, PATTERN, SYMBOL, and AXIS. All those spiffy axis labels and symbols in SAS/GRAPH displays don't just happen (when they *do* happen, it's only because you got lucky with the defaults, which, shall we say, were not subcontracted to the Apple Computer graphic design team). These statements provide control over how points are connected in a plot, whether to produce confidence intervals, how to fill in chart and other polygons, what symbol to display for a point, whether to label a displayed point with text, and the like. Just as it's unlikely that final output ready for a client would be produced by PROC PRINT, it is also doubtful that professional-looking graphics output can be produced without one or more of these statements.

FORMAT and LABEL. The use of these statements in report writing is typically that of an override. When either of these statements is used with a call to a procedure, it will be in place only for the duration of the procedure. So, if REPORT processes a data set that has a variable stored with a format of DATE9. and a FORMAT statement uses MMDDYY8. for that variable,

REPORT will [1] use MMDDYY8. to override the default value (DATE9., stored in the data set) and [2] *not* modify the format stored in the data set (it remains DATE9.).

ODS

The Output Delivery System (ODS) is possibly the most significant addition to the SAS System in the last decade. Its ability to adapt traditional SAS plain text output to a variety of popular file types – HTML, RTF, PDF, and others – provides many possibilities for the report designer. It also poses a wealth of "learning opportunities" for the programmer. The addition of file-specific control characters and sequences, a process known as "rendering," requires you to be conversant to some degree in:

- **Target file format.** You don't have to read the W3C's HTML4 specification to write an HTML file. Nor do you have to master the Adode PDF specification document to create a PDF. That's the beauty of the ODS syntax. With a single program statement, you can create an HTML frameset, a PDF, and other attractive documents. But when the need arises for a variable to be broken into multiple lines in a PDF or for hyperlinks and within-variable color-coding in an HTML document, you have to be able to modify the data being fed to the reporting procedure.
- **Target file limitations and opportunities.** Each type of file has advantages when compared to plain text or other formats. An RTF file produced by the ODS can be easily linked into a Word master document and retain the rendering – font, point size, etc. – assigned by SAS. PDF-linked master documents in Word are much more complex. Hyperlinks are easily defined in HTML, much more problematic in PDFs.
  To program for a particular target file format, you must know not only the syntax for the reporting procedure, but also the quirks for that file format-procedure combination. For example, the PROC REPORT option PANELS creates multi-column reports in plain text output but not other formats. In this example, SAS will display a note: "The option PANELS is not supported for mark-up languages such as HTML.".
  Other times, ODS-capable PROCs are not as informative. Using the WIDTH option in REPORT's DEFINE statement has no effect *and* is not flagged with a NOTE if you are writing to anything but plain text output.
  Bottom line: each type of output allows you to do things that you can't do elsewhere, and each PROC reacts a bit differently to the destinations. Read the procedure documentation carefully, review the SAS Log for notes and warnings, and verify that the output is displayed correctly.
- **PROC TEMPLATE (styles, tagsets).** The default ODS rendering for most procedures and destinations may be acceptable for some reports. It's almost inevitable, however, that regulatory requirements, client requests, or your personal distaste for a particular font will force your introduction to styles, tagsets, and templates.
  Their role in rendering is best understood if you view SAS output as a predictable arrangement of items in a display; there are titles, footnotes, BY-lines, table column headers, data in a table, etc. Once you know the names of these output elements, you can modify the default style to satisfy the display's special demands. The modification can be as minor as a change to Courier 8-point font. Modifications can also be significant, redefining virtually every element of the display.
  Styles use the terminology of object-oriented programming. Understanding how features can be inherited from a parent style is important, as is the concept of the ODS style search path (not unlike format and autocall macro search paths, but implemented with ODS-specific syntax). Note, too, that some procedures – PRINT, REPORT, and TABULATE among them – accept direct specification of style overrides, thus relieving you of the need to write an entirely new style definition with the TEMPLATE procedure.
  Tagsets are conceptually similar to styles, and even more flexible, since they can exert even more control over output rendering and are sensitive to "events" in the data. Like

styles and templates, they are a small world unto themselves, useful to know about, but usually beyond the needs of the neophyte SAS report program developer.

- **Document management (PROC DOCUMENT).** The common conception of procedure output is that it is an object (JPG, PDF, LST, etc.) written to a file in a directory. The ODS destination DOCUMENT and the tool to manage this destination, PROC DOCUMENT, give you the ability to store procedure output as device-independent objects and display them in the order and format of your choice. You can, for example, run TABULATE using BY groups, then execute REPORT. Using PROC DOCUMENT, you can create a PDF with the REPORT output displayed first, followed by some or all of the TABULATE's BY group output pieces. Since the original output is stored in much the same way as a graphics catalog or a SAS data set, the output can be rendered and reorganized multiple times without having to rerun the original procedures.

Examples

Define a template "item store" with the style NDARPT. This style inherits the properties of the style SASDOCPRINTER, overriding four properties of the SYSTITLESANDFOOTERS style. In practice, it's likely that there would be multiple style overrides required to bring the template into full comformity with the output's rendering needs.

```
libname template 'j:\common\lib';
proc template;
     define style Style.NDArpt / store=template.templates;
     parent=styles.sasdocprinter;
     /* Controls system page title text and system page footer text */
     style SysTitlesAndFooters from Container   /  Rules=groups
                                                   Frame=VOID
                                                   FrameBorder=off
                                                   BorderWidth=2
                                                   ;
     end;
run;
```

Use the template created in the previous example. Create a PDF with two user-supplied bookmarks to assist document navigation.

```
libname template 'j:\common\lib';
ods path template.templates sasuser.templat(update)
         sashelp.tmplmst(read)

ods pdf file='statusRept.pdf' style=style.NDArpt;
ods proclabel "All Patients";
proc 1
ods proclabel "Early Terminations";
proc 2
ods pdf close;
```

Render the output of several procedures as an HTML frameset. The user would typically open STATUSREPT.HTML, which would divide the browser window into a navigation window (STATUSREPT_CONT.HTML) and a results window (STATUSREPT_BODY.HTML). To ensure that table header rows are repeated at the top of the page when STATUSREPT_BODY.-HTML is printed, we specify some unintuitive and nasty-looked HTML tags via the HEADTEXT option.

```
ods html style=style.NDArpt
    frame='statusRept.html'
    body='statusRept_body.html'
    contents='statusRept_cont.html'
    headText="<style>thead {display:table-header-group} </style>"
    ;
procs go here
ods html close;
```

Macro Language

The last tool in the reporting tool box is the all-encompasing SAS macro language. Think of macros as pieces of code that [1] execute prior to the PROC or DATA step that precede or contain them and [2] can generate some, all, or multiple statements for SAS to execute. The range of macro applications is vast, and their penetration into every level of SAS programming is significant. It's hard to imagine calling yourself a SAS programmer without having at least a nodding acquaintance with macro capabilities and syntax.

From the point of view of report writing, let's identify some of the activities that the macro language can perform:

- Set page orientation and margins, based on the value of a macro parameter (e.g., ORIENT=PORTRAIT).
- Count the number of observations in a data set. If zero, write a message to the output destination, otherwise print a report.
- Run diagnostic PRINT and FREQ procedures, based on the value of a macro parameter (e.g., DEBUG=YES). Conditional execution of the procedures is helpful during the report's development and later, for debugging or enhancing.
- Based on the output destination, insert specific ODS commands. HTML output specification should include table header commands, but specifying these for the PDF destination would create an error.
- For the HTML destination, insert mark-up characters around a variable. If, for example, a variable is not within a range of values, a new variable can be created, rendering the value as bold and red. Rather than continually repeat the coding sequence for the HTML tags, it is easier to write a macro that can be used by multiple programs.

If the piecemeal, divide-and-conquer approach to program development was viewed as "nice to have" earlier in this paper, it becomes a "must have" when writing macro code. Build and test the program in pieces. First, ensure that any starting values (macro parameters) are valid. Then build the framework. If, for example, the macro had to print from all data sets in a library, you would first build a list of all the data sets. A well-designed macro should react one way to no data sets found and another, happier way to one or more that were located. The %IF - %THEN sequences of the macro roughly mimic your questions and reactions ("I built a list of data sets. Is it empty? If so, write a message and leave. Otherwise, loop once for every data set in the list.")

Examples

While macro DATAN is not a reporting tool *per se*, it is presented here to illustrate an important point about reporting-writing macros, and macros in general. Rather than continually write macro code to count the number of observations in a data set, check for the presence of a variable in a data set, create a list of data sets in a library, and so on, the macro developer should be able to utilize a library of macros that perform such tasks. This speeds development, and has the added benefit of the program being reliant on validated utilities. In the case below, we pass the name of a data set and DATAN returns macro variable COUNT (-1 if the data set was not found, 0 or greater otherwise).

```
%macro dataN(data=);
       %global count;
       %let dsid  = %sysfunc(open(sasuser.&data.));
       %if &dsid. > 0 %then %do;
           %let count = %left(%sysfunc(attrn(&dsid., nObs)));
           %let rc    = %sysfunc(close(&dsid.));
           %end;
           %else %let count = -1;
%mend;
```

The RPT macro that follows is a simple program shell demonstrating a number of features that make macro programming powerful and appealing:

- Only parameters need to be changed to produce different results. The macro code itself remains untouched.
- Low-level macros do the work when possible (use of DATAN).
- Conditions are tested as the macro executes, preempting execution of code that would create an error (we terminate if the data set is not found)
- Page margins are set by the macro. All the user needs to do is specify the orientation.
- Output can be tailored on the fly, based on data set size (there is one form of output for empty input data sets, another for data sets with one or more observations).

The macro is far from complete; there is no parameter checking, for example. Yet even this rough outline suggests the power and flexibility of the language.

```
%macro rpt(orient=portrait, data=, draft=yes);
       %dataN(data=&data.);
       %if &count. = -1 %then %do;
           print data set not found message, then terminate
           %end;

       %if %upcase(&orient.) = PORTRAIT %then %do;
           set ORIENTATION, TOP/BOTTOM/RIGHT/LEFT margin options
               needed for portrait
           %end;
           %else %do;
                 set ORIENTATION, TOP/BOTTOM/RIGHT/LEFT margin options
                     needed for landscape
                 %end;

        %if &count. = 0 %then %do;
            write "no obs in input data set" message to output
            destination
            %end;
            %else %do;
                  statements for PROC REPORT
                  %if %upcase(&draft. = YES) %then %do;
                      footnote2 "*** DRAFT *** DRAFT *** DRAFT*** DRAFT*** DRAFT";
                      %end;
                  %end;
%mend rpt;
```

Conclusion

We've just taken a long tour at a high altitude. The breadth of our discussion required that we bypass syntax, knowing that this is handled well in SUGI papers, books, and the online documentation. Emphasized here are the matters most important to our target, novice SAS user audience:

- Carefully examine report specifications for what they say and what they omit
- Be aware of the demands imposed by different types of output
- Become familiar with available macros, templates, and program code base
- View the reporting process as a series of steps, not just a call to a single procedure
- Become fluent in the reporting tools you need, but become conversant in all the tools at your disposal.

Your comments and questions are always welcome. Contact the author at Frank@CodeCraftersInc.com.