

A Flexible System of Reporting Safety Data in Clinical Trials
Gregory C. Steffens G.D. Searle

Reporting, from the programmer's perspective, is the end-product of a clinical study. From a broader perspective, reports are used to make strategic decisions about the efficacy and safety of a test article. Reporting therefore is a highly critical milestone in a clinical study. The programmer's and designer's task is to design a system that reports data with an extremely high degree of reliability. This design task is all the more challenging because the system must also deliver reports with great speed and flexibility in a research environment which, by its nature, presents a varying set of database structures and data points to report. Standardization of these is an admirable goal and should be pursued, but will never fully be realized since research means the questions to be asked will change and thus the data collected to answer these questions will change as well. These issues are important in clinical studies as well as any reporting or decision-support applications. The goal of quickly producing reliable reports of datasets that can change in structure is a goal common to many different industries and environments.

To meet this goal a programmer should conform to a design principle of defining program modules that are data-driven and re-usable for all the varieties of database it needs to report or manipulate. The antithesis of this principle is what I call the

"tweak-and-run" principle, where a programmer copies a report program for a similar database to his/her own work area, modifies ("tweaks") the program for the new database and runs the modified program. This is neither quick nor reliable, since programmer intervention is required for every new database (creating a backlog of studies waiting to be reported) and not offering any single program module that can be validated. It is very resource intensive for the programmer's time and the time it takes the data monitors to submit reports to "validation-in-use"; i.e. fully check every report every time it is run. The better design principle is what I call the utility principle where a programmer creates utility modules that are reusable and data-driven. Creating reusable data-driven modules will take more time to develop than any single "tweak-and-run" program, but in the long run it saves time and increases quality. The reusable modules can be validated when created. These modules can then be used as a simple "black-box" making highly complex tasks a simple one-line call to the module. An added benefit, especially in an environment where outside contract programmers are used, is that the reusable modules can enforce corporate policies about how data is to be manipulated and calculated.

The SAS System itself is an example of this utility

principle. A simple example is the "lowly" PROC PRINT. This procedure will print variables from a SAS dataset and allow simple customizations to be done - a much easier task than writing a PUT statement, a FORTRAN WRITE or an assembler MVC! PROCs TABULATE and REPORT carry the customization possibilities of this print utility further. A much more sophisticated and excellent example of the utility principle is the new SAS/PH-Clinical¹ product. This is an extremely flexible data-driven system allowing non-programmers to report and view clinical study data themselves. That SAS/PH-Clinical adheres to the utility principle, in my opinion, puts this product in a class by itself.

The macro language in the SAS System is a very strong ally in the effort to adhere to the utility principle. Another ally is the meta-dataset; i.e. a SAS dataset whose variables contain descriptive information about study databases. SAS macros can be written which read the meta-datasets and the datasets to be reported, place certain information in macro variables and use this information to generate SAS code that is customized to the database to be reported. The customization is flexible to the varying database structures. But it is also flexible to the different instantiations of a report format. For example, a report may contain one column for each treatment period in a study but different studies will have a differing number of treatment periods causing a different

number of columns in various runs of the report program. This, with the fact that the number of rows will vary as well, requires the macro to be quite "intelligent" about the nature of the study database it reports as well as how to summarize and report the data. This intelligence is contained mostly in the macro code and the meta-datasets. To see more clearly what this intelligence is, some examples will prove helpful. So, we will look at reporting signs and symptoms experienced by a subject during participation in a clinical trial. Subjects will be asked to describe all the symptoms they have experienced from the time of their previous visit to the time of their current visit including the dates and times the symptom started and ended. From these dates and times the duration of the symptom can be calculated. Some studies require this duration to be calculated to the minute while other studies will require that the duration be calculated only to the number of days. Therefore, some studies will collect the times of start and stop while others will collect only the dates but not the times. A SAS program which references a time variable willabend when it is run against the study that does not collect time.

So, how can one program be intelligent enough to report the durations of symptoms in both kinds of studies? To start, we create a meta-dataset containing report parameters - name this dataset REPTPARM. This dataset has one observation per study and along with other variables it

contains four variables whose values are the names of the variables in the study symptoms dataset which hold that start date, start time, end date and end time; STARDATE, STARTIME, ENDDATE and ENDTIME, respectively. For one study the STARTIME and ENDTIME will be missing in REPTPARM. For another study these same variables will have non-missing values. Both studies, we will assume, have values in STARDATE and ENDDATE. We can take advantage of this bit of intelligence with SAS macro language. We SET REPTPARM, call SYMPUT to place the start and end variable names into macro variables and use %IF... %THEN... to generate SAS code customized to whether or not the study dataset contains time variables. If the REPTPARM variables are missing, the macro will not generate SAS code which references the time variables. If the REPTPARM variables are not missing, the macro will generate SAS code that references time variables and references them by the variable names specified in REPTPARM. This is a simple example, but the principle is important: the meta-dataset can tell a SAS macro what a database looks like, the macro code will use this information in deciding what SAS code to generate and the resultant SAS code is customized to the database being reported. The principle can be easily extended beyond specifying dataset and variable names - the meta-dataset can indicate what kind of subsetting to perform or any other required customization.

The generated SAS code itself is also responsible for intelligence

and flexibility. When the PUT statement produces a report of the contents of a dataset, the program should first read through the dataset to determine the width of columns and the starting position of each column based upon these widths. One study may require 40 positions to print the description of the symptom, but another study may require only 20 positions. SYMPUT can be used to place each column's maximum width and the PUT statement can use that when specifying the width in the character format. The report program should also compute the total width required to print all the columns so that the linesize and position of the first column can be computed to center the report on the page and decide if the report requires portrait or landscape orientation. If the PUT statement is included in a report macro, end-user customizations can be made possible with the use of macro parameters. Columns can be turned on or off, sort order specified, the number of blank spaces between columns can be set by the user and any other customization that is of enough benefit to program into the macro.

Once the system of macros is created, an upper layer can be placed over the macros to allow end-users to set the values of macro parameters that relate to the customization functions of the report macros. This upper layer will allow users to "tell" the macros which study database to report, what subset of that database to report, which columns to include on the report, etc. This layer between the user and

the system of macros is best written in SAS/AF² and SAS/FSP³. The AF screens will prompt the user for macro parameter values, edit the response, build a SAS program and submit the program for execution in either interactive or batch mode.

When designing a system of macros, one must evaluate when to split parts of the system into different pieces. How does one decide how many macros make up a system? One primary rule, according to the utility principle, is that if a functional unit is common to more than one program, it should be split off into a separate macro. A second layer of macros now emerges. Some functions performed by macros will be so shareable and fundamental that they will be called by other macros. As one delves into deeper layers of the macro system, the macros become less and less specific to any one application or system of macros. We end up with a many-layered system with SAS/AF at the uppermost layer having the most direct contact with the world outside the macro system. One layer down are the macros which directly manipulate study datasets and print reports. Layers under these macros are the macros that serve other macros in the layers above them. The macros in these lower layers conform more and more purely to the utility principle. They become more and more re-usable and reliable since any "bugs" in the deeper layers that survive validation will be more likely to be made apparent because they are called by many modules in the

upper layers. Under the bottom layer of your system lies the layers that make up the SAS System multiple vendor architecture (MVA) layers. Again the SAS System proves to be an excellent example of the utility design principle, this time in its use of the MVA which insulates the kernel of the SAS system from the many different operating system environments in which it runs. The end-user is less directly aware of the deeper layers but benefits immensely in that the overall reliability and flexibility of the system is greatly increased by the utility principle. The programmer benefits because the existence of utility macros make complex jobs a simple macro call and allows a faster development of programs. The company benefits because reliable reports can be generated more quickly using "canned" algorithms that enforce corporate policies regarding how certain data manipulations and calculations should be performed.

-
1. SAS/PH-Clinical is the trademark of SAS Institute, Inc., Cary, NC, USA
 2. SAS/AF is the registered trademark of SAS Institute, Inc., Cary, NC, USA.
 3. SAS/FSP is the registered trademark of SAS Institute, Inc., Cary, NC, USA.

```

%MACRO SAFEINC(DATA=,STUDY=,START=,STOP=,BODYSYS1=Y,BODYSYS2=N,BODYSYS3=N,PROTOCOL=);
OPTIONS CC=CR;
%LOGICAL(BODYSYS1)
%LOGICAL(BODYSYS2)
%LOGICAL(BODYSYS3)
%IF &DATA = %THEN %LET DATA = &DSNOUT;
%IF &BODYSYS1 %THEN %LET BODSYS = PRIMARY;
%IF &BODYSYS2 & ^ &BODYSYS3 %THEN %LET BODSYS=&BODSYS AND SECONDARY;
%IF &BODYSYS2 & &BODYSYS3 %THEN %LET BODSYS = &BODSYS, SECONDARY;
%IF &BODYSYS3 %THEN %LET BODSYS = &BODSYS AND TERTIARY;
%IF %STR(&BODSYS) ^= %THEN %LET BODSYS = &BODSYS BODY SYSTEMS REPORTED;
%IF %EVAL(&BODYSYS1+&BODYSYS2+&BODYSYS3) > 1 %THEN %DO;
  DATA REPTDATA;
    SET &DATA;
    %IF &BODYSYS1 %THEN %STR(IF BODYSYS1 ^= '' THEN OUTPUT;);
    %IF &BODYSYS2 %THEN %STR(IF BODYSYS2 ^= '' THEN DO;
      BODYSYS1 = BODYSYS2; OUTPUT; END;);
    %IF &BODYSYS3 %THEN %STR(IF BODYSYS3 ^= '' THEN DO;
      BODYSYS1 = BODYSYS3; OUTPUT; END;);
  RUN;
  %LET DATA = REPTDATA;
%END;
PROC SORT DATA = &DATA (KEEP=PROTOCOL BODYSYS1 TRMT DOSE REGMN CODE SUBNO
  WHOPREF ENTDT) OUT=REPTDATA;
  BY BODYSYS1 WHOPREF PROTOCOL TRMT DOSE REGMN CODE SUBNO;
  %IF %BQUOTE(&PROTOCOL) ^= %THEN %DO;
    WHERE PROTOCOL IN (
      %LET I = 1;
      %DO %WHILE (%QSCAN(&PROTOCOL,&I,%STR( )) ^= %STR());
        "%SCAN(&PROTOCOL,&I,%STR( ))"
        %LET I = %EVAL(&I + 1);
      %END;
    );
  %END;
  .
RUN;
DATA REPTDATA;
  SET REPTDATA;
  BY BODYSYS1 WHOPREF PROTOCOL TRMT DOSE REGMN CODE SUBNO;
  IF FIRST.SUBNO THEN SUBCOUNT = 1;
  ELSE SUBCOUNT = 0;
RUN;
PROC SUMMARY DATA=REPTDATA MISSING NWAY;
  CLASS PROTOCOL;
  VAR ENTDT SUBCOUNT;
  OUTPUT OUT=PROTSMRY N=FREQ SUM(SUBCOUNT)=INCIDENC MIN(ENTDT)=STARTIME
    MAX(ENTDT)=STOPTIME;
RUN;;
PROC SUMMARY DATA=REPTDATA MISSING NWAY;
  CLASS BODYSYS1 WHOPREF PROTOCOL TRMT DOSE REGMN;
  VAR SUBCOUNT;
  OUTPUT OUT=SMRY N=FREQ SUM=INCIDENC;
RUN;
DATA _NULL_;
  IF EOF THEN DO;
    CALL SYMPUT('MAXWPLEN',PUT(MAXWPLEN,2.0));
    CALL SYMPUT('MAXTLEN',PUT(MAXTLEN,2.0));
    CALL SYMPUT('MAXDLEN',PUT(MAXDLEN,2.0));
    CALL SYMPUT('MAXRLEN',PUT(MAXRLEN,2.0));
    TOTLEN = 4 + MAXWPLEN + 18 + MAXTLEN + 1 + MAXDLEN + 1 + MAXRLEN + 1 + 4;
    CALL SYMPUT('TOTLEN',PUT(TOTLEN,3.0));
    IF TOTLEN <= 80 THEN LS = 80;
    ELSE LS = 132;
    CALL SYMPUT('LS',PUT(LS,3.0));
  END;
  SET SMRY END=EOF;

```

```

IF LENGTH(WHOPREF) > MAXWPLEN THEN MAXWPLEN = LENGTH(WHOPREF);
IF LENGTH(TRMT) > MAXTLEN THEN MAXTLEN = LENGTH(TRMT);
IF LENGTH(DOSE) > MAXDLEN THEN MAXDLEN = LENGTH(DOSE);
IF LENGTH(REGMN) > MAXRLEN THEN MAXRLEN = LENGTH(REGMN);
RETAIN MAXWPLEN MAXTLEN MAXDLEN MAXRLEN;
RUN;
OPTIONS LS = &LS;
PROC PRINT DATA = PROTSMRY (DROP= TYPE FREQ);
  TITLE 'PROTOCOLS INCLUDED IN THIS REPORT';
  %IF %EVAL(&BODSYS1+&BODSYS2+&BODSYS3) > 0 %THEN %STR(
    TITLE2 "&BODSYS");
  FORMAT STARTIME STOPTIME DATETIME14.;
RUN;
PROC SORT DATA = SMRY;
  BY BODSYS1 INCIDENC WHOPREF PROTOCOL TRMT DOSE REGMN;
RUN;
DATA _NULL_;
  SET SMRY;
  BY BODSYS1 INCIDENC WHOPREF PROTOCOL TRMT DOSE REGMN;
  IF _N_ = 1 THEN DO;
    C1 = MAX((INT(&LS - &TOTLEN) / 2),1); /* BODSYS */
    C2 = C1 + 4; /* WHOPREF */
    C3 = C2 + &MAXWPLEN + 1; /* PROTOCOL */
    C4 = C3 + 16 + 1; /* TRMT */
    C5 = C4 + &MAXTLEN + 1; /* DOSE */
    C6 = C5 + &MAXDLEN + 1; /* REGMN */
    C7 = C6 + &MAXRLEN + 1; /* INCIDENC */
    CT1 = CEIL((&LS - 27) / 2); /* BODSYS TITLE */
    CT2 = CEIL((&LS - LENGTH("&BODSYS")) / 2); /* BODSYS TITLE */
    CD = CEIL((&LS - 47) / 2); /* DATE TITLE */
  END;
  FILE PRINT HEADER=HEADER NOTITLES CC=CR PS=58 N=PS;
  IF FIRST.BODSYS1 THEN PUT / @C1 BODSYS1 /;
  IF FIRST.WHOPREF THEN PUT @C2 WHOPREF $&MAXWPLEN.. @;
  IF FIRST.PROTOCOL THEN PUT @C3 PROTOCOL $16. @;
  IF FIRST.TRMT THEN PUT @C4 TRMT $&MAXTLEN.. @;
  IF FIRST.DOSE THEN PUT @C5 DOSE $&MAXDLEN.. @;
  IF FIRST.REGMN THEN PUT @C6 REGMN $&MAXRLEN.. @;
  PUT @C7 INCIDENC 4.0;
  RETURN;
  HEADER:
  IF _N_ = 1 THEN DT = DATETIME();
  PAGENO + 1;
  PUT #1 'SAFEINC.SAS' @20 DT @73 'PAGE ' PAGENO 3.0
    #8 @CT1 'INCIDENCE OF ADVERSE EVENTS'
    %IF %EVAL(&BODSYS1+&BODSYS2+&BODSYS3) > 0 %THEN %STR(
      // @CT2 "&BODSYS");
    %IF &START ^= &STOP ^= %THEN %STR(
      // @CD "FOR RECORDS ENTERED BETWEEN &START AND &STOP");
  /;
  RETURN;
  FORMAT DT DATETIME18.;
  RETAIN C1-C7 CD CT1 CT2 DT;
RUN;
%MEND;

```