Using Pharmacovigilance Reporting System to Generate Ad-hoc Reports

Jeff Cai, Amylin Pharmaceuticals, Inc., San Diego, CA
Jay Zhou, Amylin Pharmaceuticals, Inc., San Diego, CA

ABSTRACT
To supplement Oracle Adverse Event Reporting System, many SAS macros are implemented to enhance data cleaning, pharmacovigilance reporting, regulatory submissions, and other surveillance activities. These macro programs have been defined to the point that only simple parameter changes are needed to meet business needs.

An SAS AF application, Pharmacovigilance Reporting System, is developed to save programming resources and allow data analysts with less or no SAS programming skills to generate the routine ad-hoc reports. With a few mouse clicks and inputs, users can easily subset, analyze, and present safety data with standard reports. This paper will review and discuss the functionalities of the application in detail.

INTRODUCTION
To take advantage of the powerful analysis and reporting capability with SAS, many reports generated by SAS have been used to support case management, safety monitoring, management reviews, and regulatory submissions.

In addition to routine reports, ad-hoc sub-group analyses are often requested to examine the data. For example, a number of reports with the same reporting template can be generated based on the different subset data such as serious cases only, United States cases only, or cases with some specific adverse event preferred terms. To meet such challenges, a set of parameters need to be added to the macro programs for easy maintenance. However, it is tedious to maintain the programs calling the macros, especially if there are many programs, which is not be the best use of programming resources.

To save programming resources and allow those safety data analysts who have few or absent SAS programming skills to generate routine ad-hoc reports, an SAS AF application, Pharmacovigilance Reporting System (PVRS) was developed. The safety data analysts can directly generate SAS reports from the system without asking for help from SAS programmers. This will help them to better control their reporting schedules.

APPLICATION OVERVIEW
The PVRS was developed using SAS/AF® software, which is a set of tools to construct a visual application. Drag-and-drop components in SAS/AF® frame development environment are used to build the Graphical User Interface (GUI). SAS Component Language (SCL), the scripting language behind SAS/AF®, is coded to facilitate this interactive application. Using SCL, the application will be able to generate and submit SAS source code. Because SAS/AF® applications are stored in SAS catalogs, they are portable to all SAS software platforms. After the application is compiled, it can be accessed by users who have licensed Base SAS® software installed on their computers.

The main interface of PVRS is shown in Figure 1. We use red boxes to divide the interface into five sections to describe the functions within the system. Figure 2 is the system processing flowchart reflecting the system architecture and the processing flow.
Figure 1. The main interface of PVRS

![Main Interface of PVRS](image)

Figure 2. The Process Flowchart

![Process Flowchart](image)
MAINTAINING THE REPORT LIST

Once the PVRS starts, the system will select all available report templates from a dataset and populate those in the window in Section A of Figure 1. When the system is initialized, all related macros are activated and data libraries are set up. The macros are the cornerstones of the PVRS. Through the management of PV report specifications, data analysts know the usage of these macros. The PVRS provides the interface to accept values to feed parameters in the macro, and then execute the macro to generate the pharmacovigilance (PV) reports.

In order to set up the link between the report templates in the PVRS and the corresponding macros, the PVRS provides an interface shown in Figure 3 to manage macro information such as macro names, parameter names, and default values (for some parameters). Using this interface, data are captured and maintained in an SAS data set. Without changing any code of the PVRS, the number of report templates can be easily expanded, thus increasing the reporting.

With the same principal, similar interfaces are also provided to set up and maintain all the other configuration parameters of the PVRS. All these maintenance interfaces are developed with the FSEDIT procedure in SAS/AF®.

Figure 3. An Interface for Report Setup and Maintenance

ASSEMBLING THE MACRO CALL

Users can use controls in section B of Figure 2 to configure the parameters that we discussed at the beginning of this paper. If a macro program has more than these nine parameters, the PVRS will get the remaining parameters and default values from a data set which is managed by the interface of Figure 3. The users can configure multiple reports and manage their execution order in the window of Section C of Figure 4.

We use SCL lists as the data structure to temporarily store and manipulate data inside the PVRS since the SCL lists are dynamic, i.e., they can grow and shrink to accommodate the number of items assigned to them. The list size is computed during run time rather than during compile time. An SCL list can contain items of different data types including numeric data, character data, or even another SCL list. Figure 5 shows the data structure and illustrates how to assemble these data into a collection of macro calls.
As long as a report is selected in the PVRS, its corresponding macro name will be retrieved from a data set and be added to a macro SCL list. The parameters corresponding to each macro will also be added to an SCL list. Then this list, as a new item, will be added to a parameter SCL list. With the same principal, values collected from the interface will also be added to a list in the same order as the parameter list. The first line in Figure 5 shows an example of a macro SCL list. \%pv_001 and \%pv_002 are two macro names stored in the list. The second and the third line show the parameter SCL list and the value SCL list, respectively. Each value will match with the corresponding parameter name so that we can assemble all the data in these SCL lists into macro calls. In this way, all resources related with one report will be assembled to one macro call. The
PVRS will use the SUBMIT CONTINUE statement to submit each macro call to Base SAS for execution.

**SUBSETTING FUNCTION**

In order to generate ad-hoc reports with a flexible data manipulation tool, the PVRS provides a critical subsetting function to allow users to analyze the cases of interest. Figure 6 shows the subsetting interface and an interactive WHERE window. The user can choose to build either case level subsetting criteria or event level criteria, or both. A case level subsetting selects all qualified case IDs as the analysis population, while an event level subsetting chooses all combination of qualified case ID and event ID as the analysis population. The PVRS will call the _setWhere method to activate the interactive WHERE window to specify a WHERE expression. Then, another data set data model method, _getWhere, will be called to send the WHERE expression to a SCL list. Furthermore, the WHERE expression will be transformed into a string and be assigned to the macro parameter whereExp=. In each report program, the macro %subsetting will be called to decode this string into multiple WHERE statements to accomplish the subsetting task. The code of %subsetting will be shown as an Appendix.

**SYSTEM OUTPUTS**

Besides the PV reports, the PVRS also produces an application log and a report log for each report during the program execution. The system will scan each report log to decide the execution status. If there is an error message, an error notification will be emailed to the responsible analyst. The source codes for the log scan function and email function will be attached at the Appendix.

The PVRS can save the reporting history. For each reporting, information such as the report name, macro call, report execution duration, execution status, user name, and reporting date and time will be recorded in a reporting history dataset. The PVRS provides functions to search and display reporting history data. The user can also choose to export history data to an external Excel spreadsheet.

To improve the efficiency of the reporting and save the report configuration, the PVRS also provides a function to export the configured data to the SCL lists (Figure 5) in a portable SAS catalog. We name the catalog as a PVRS project (Figure 2). Later, the user can load the PVRS
project into the system to start execution directly without any data entry or only with modifications. This function can allow users to build up their customized reporting package library.

APPLICATION DEPLOYMENT AND SECURITY
To start a SAS session, we create an icon to allow users to have access to the SAS executable file and to a customized SAS configuration file. A standard SAS configuration file is supplied with SAS software and is typically located in the same directory as the SAS executable file. We modify some options in this standard configuration file to meet our security requirement as in the following:

```sql
/*Setup options AF application*/
-set Rptmgt "C:\PharmaSUG2008"
-awscontrol title nosystemmenu minmax
-awstitle 'Pharmacovigilance Reporting System (Version 1.0)'
-noawsmenumberge
/*-sascontrol nosystemmenu nominmax*/
-nolog
-nolog
/*-noawsmenu*/
-initcmd "af c=Rptmgt.reportmanagement.Srmain.frame pmenu=YES;
  toolclose; command close; wstatusln off;"
```

With these options, the application workspace (AWS) menu will be suppressed. SAS tools, command box, SAS log window, and even SAS logo will be closed. The PVRS, with its customized application menu (Section E in Figure 1), takes the place of the SAS menu. This configuration options limits the user to use only the functions provided by the PVRS.

CONCLUSION
The PVRS provided a powerful tool for users without SAS programming experience to generate ad-hoc PV reports. The system was easy to use and maintain. With a few clicks and inputs, users can quickly generate the reports in a consistent layout. Therefore, it greatly facilitated the realization of their daily tasks.

ACKNOWLEDGEMENTS
We would like to thank David Clawson for his encouragement and review of this paper.

SAS and all other SAS Institute, Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

CONTACT INFORMATION
Jeff Cai
Amylin Pharmaceuticals, Inc.
9360 Towne Centre Drive, Ste. 110
San Diego, California 92121
Jeff.Cai@amylin.com
APPENDIX

1) Macro %subsetting

```sas
%macro subsetting(dsnin=,dsnout=,wh=,level=case);
*detect if case_id is in dataset &dsnin;
%let dsid=%sysfunc(open(&dsnin,i));
%if &dsid=0 %then %do;
  %put;
  %put ERROR: Dataset &dsnin can not be opened;
  %put %sysfunc(sysmsg());
  %put;
  %goto EndMacro;
%end;
%let varCk=%sysfunc(varnum(&dsid,case_id));
%if &varCk=0 %then %do;
  %put ;
  %put ERROR: Dataset &dsnin has no variable case_id. Program is
terminated!;
  %put ;
  %let rc=%sysfunc(close(&dsid));
  %goto EndMacro;
%end;
%let rc=%sysfunc(close(&dsid));
%if %length(&wh) %then %do;
  %if %upcase(&level)=EVENT %then %do;
    *detect if event_seq_nbr is in dataset &dsnin;
    %let dsid=%sysfunc(open(&dsnin,i));
    %if &dsid=0 %then %do;
      %put;
      %put ERROR: Dataset &dsnin can not be opened;
      %put %sysfunc(sysmsg());
      %put;
      %goto EndMacro;
    %end;
    %let varCk=%sysfunc(varnum(&dsid,event_seq_nbr));
    %if &varCk=0 %then %do;
      %put ;
      %put ERROR: Dataset &dsnin has no variable event_seq_nbr. Can not do
      event level subsetting!;
      %put ;
      %let rc=%sysfunc(close(&dsid));
      %goto EndMacro;
    %end;
    %let rc=%sysfunc(close(&dsid));
    %let sby_=case_id event_seq_nbr;
  %end;
  %else %let sby_=case_id;
  proc sort data=&dsnin out=&dsnout;
    by &sby_;
  run;
*subset each dataset;
%let wh_=compress(&wh, \); %let whN_=eval(%length(&wh) - %length(&wh_) + 1);
%do j=1 %to &whN_; %let eachWh_=scan(&wh, &j, \);
  %let pSet_=scan(&eachWh_,1,@);
  %let process_=scan(&eachWh_,2,@);
```
*detect if event_seq_nbr is in dataset &pSet_;
%let dsid=%sysfunc(open(&pSet_,i));
%if &dsid=0 %then %do;
  %put;
  %put ERROR: Dataset &pSet_ can not be opened;
  %put %sysfunc(sysmsg());
  %put;
  %goto EndMacro;
%end;
%let varCk=%sysfunc(varnum(&dsid,event_seq_nbr));
%let rc=%sysfunc(close(&dsid));
%if &varCk=0 %then %do;
  *if event_seq_nbr is not in dataset, case_id is selected only for
  subsetting;
  proc sql noprint;
    create table &dsnout as
    select * from &dsnout
    where case_id in (select case_id from &pSet_(where=(&process_)));
  quit;
  %end;
%else %do;
  proc sort data=&pSet_(where=(&process_)) out=temp_(keep=&sby_) nodupkey;by &sby_;run;
  data &dsnout;
    merge &dsnout(in=a) temp_(in=b);by &sby_;
    if a and b;
  run;
  %end;
%end;
%else %put No input;
%EndMacro:  %put ***** End of MACRO Subsetting*****;
%mend;

2) Macro %scanLog

%macro scanLog(logPath=);
  %global _logSig;
  %let sig_err=;
  %let sig_pro=;
  %let sig_war=;
  /* Read log file.; */
  filename InLog "&logPath";
  data LogFile Findings;
    length inputLine $256;
    infile InLog truncover;
    input @1 inputLine $char256.;
    output LogFile;
  /*Store wanted message to print at bottom of log file.; */
  if ( substr(inputLine, 1, 7) eq 'WARNING' or
       substr(inputLine, 1, 5) eq 'ERROR' or
       substr(inputLine, 1, 7) eq 'PROBLEM' )
    then do;
      linenumber = _n_;  
      Finding = 'Found At Line:' ||put(linenumber, z6.)||' - '||inputLine;
      output Findings;
      if substr(inputLine, 1, 5) eq 'ERROR' then
        call symput('sig_err','ERROR')
      else if substr(inputLine, 1, 7) eq 'PROBLEM' then

call symput('sig_pro','PROBLEM');
else if substr(inputLine, 1, 7) eq 'WARNING' then
call symput('sig_war','WARNING');
end;
run;
%if &sig_err=ERROR %then %let _logSig=ERROR;
%else %if &sig_pro=PROBLEM %then %let _logSig=PROBLEM;
%else %if &sig_war=WARNING %then %let _logSig=WARNING;
%else %let _logSig=SUCCESS;
/*================================================================*
* Rewrite file and append any findings at bottom of log.*
filename OutLog "&LogPath";
data _null_;    
  file OutLog notitle recfm=V lrecl=256;
  set LogFile(in=a) Findings(in=b);
  if a then put inputLine;
  else do;
    put ' ';
    put Finding;
    put 'Analyst Comments:';
    put ' ';
  end;
run;
%mend;

3) Macro %af_email

%macro af_email(fdir=,emailTo='Jeff.Cai',emailCC=);
  %if %length(&emailCC) %then
    %str(filename mymail email to=&emailTo cc=&emailCC type='text/html');
  %else %str(filename mymail email to=&emailTo type='text/html');
subject="Notification of AF application running errors"
/*attach="&fdir"*/;
data _null_;  
  file mymail;
  put "This email is to notify you to open log";
  put "<b><A href="file:///\&fdir">&fdir</A></b> to analysis application running errors";
  put ' ';
run;
%mend;