A Non-Invasive Macro to Track Submission Metadata in SAS Drug Development

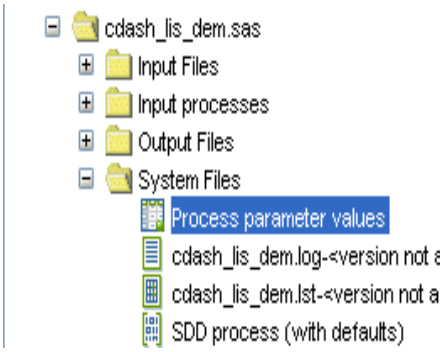
Katrina E. Canonizado, Celerion, Lincoln, NE
Bradford J. Danner, i3 Statprobe, Lincoln, NE
Matthew J. Wiedel, Celerion, Lincoln, NE

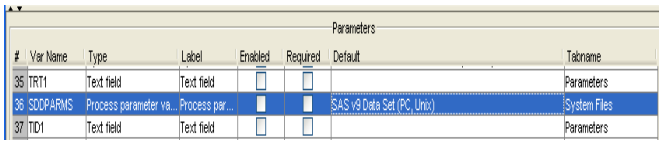
ABSTRACT

To address the concerns of program quality and transparency in the contract research organization (CRO) industry, program validation plays a central role in data analysis. As an aid to speed up this process, we provide fellow programmers and reviewers with what we refer as an audit program list. This list contains the current settings of the SAS® session like the program name, data sets used, etc. In SAS Drug Development, this information is stored in SDDparms for individual programs within submission. This paper demonstrates briefly the macro %metadata we have written. The macro primarily utilizes SDDparms in tracking the information to build the audit program list.

INTRODUCTION AND BACKGROUND

An audit program list is what we refer to as a compilation of program names, table and figure titles included in the clinical study report, permanent datasets used within the program, rundate and runtime, and macros called within individual programs. This is a part of our program documentation, if requested, submitted to reviewers to ensure the traceability, reliability and accuracy of the programs and of the outputs that go with every study report. There are several ways to gather these relevant program components using basic SAS functions such as PROC CONTENTS, SAS Dictionary Tables, use of reserved SAS macro variables like &sysdate, &systime and &sysdsn. However, in the SAS Drug Development (SDD) environment, within every run cycle, there is an optional parameters dataset available to the user and is accessible through a parameter list as illustrated below. This dataset, which contains many of the items needed to create an audit program list, can be accessed programmatically using the global macro variable &SDDPARMS.





Most of the standard programs we have, used the information within the SDDParms dataset for a specific purpose. With the increasing demand of process and program validation demanded by the industry, there was a need to revisit our current programs and available resources. One of our biggest challenges was how to access resources in a non-invasive way within our existing table and figure programs. Because there are several programs that are currently being used, our goal was to include a feature within those programs with the capability of capturing the information required for an audit program list without recoding the programs. To help us with this, we developed a macro called %metadata.

The key procedure used in the %metadata macro is PROC SQL. PROC SQL is an effective tool for creating, modifying, and retrieving data dynamically in relational databases. Moreover, PROC SQL has the capability of creating macro variables in such a way that the user can:

- create several macro variables with a shorter statement;
- trace easily the parameters or conditions being set;
- store a list and grab values in a macro variable without having to manually enter; and
- combine BASE SAS tools without interfering with the initialization of the macro variables.

METHODS AND DISCUSSION

The %metadata macro is invoked at the very end of every table, listing and figure program within a clinical study report submission. The macro is activated when the output are being finalized to minimize rework and to keep the most accurate record.

Since all the table and figure programs within a submission are calling the macro, it would be tedious to alter each individual program to activate this macro. To avoid too much work, the %metadata macro is compiled within a central setup program, which contains certain processes, macros and formats necessary for all the TFL programs, and is already included in all the standard programs, therefore, the %metadata macro is available to be called by the user anytime it is needed.

The code begins with a %LET statement to toggle the running of the macro. While trying to set-up the TFL programs, initially the macro variable &choice is set to NO (Refer to Appendix). Because there will be continuous updates and changes in production programs included in a submission, it is a good decision not to activate this macro until the tables and figures have been approved. When there is indication that the TFLs are finalized, the %metadata macro is actived by setting the &choice variable to “YES”, and information necessary for the program audit list is compiled by re-running all the programs.

Most of the information compiled by the macro is accomplished by filtering for relevant record contained in the SDDPARMS dataset and assigning the values to macro variables using PROC SQL. To create macro variables using PROC SQL, the syntax is:

```
SELECT <variable name in a SAS data set>

INTO :<macro variable name>

(and optionally)

 SEPARATED BY '<a blank, a character, or a character string>'
```

Following this format and with a little manipulation using the WHERE statement, as shown below, we were able to retrieve the program name, programmatically using the global macro variable &SDDPARMS:

```
proc sql noprint;
select value into :mtdpnam from &SDDPARMS
   where (id="<process>" and valtype="filename");
```

#	Parameter Level	Parameter ID	Parameter Label	Parameter Type	Value Type	Value	Update
1		<run>	Run information		date	2011-01-25T20:07:20	
2		<run>	Run information		user	canonik01	
3	<MAIN>	<process>	SAS Process		filename	pc-pk-conc-tables-sig.sas	
4	<MAIN>	<process>	SAS Process		path	/MDS/development/users/ECR/kc9/trial/pksas	
5	<MAIN>	<process>	SAS Process		version	<version not available>	
6	<MAIN>	<process>	SAS Process		system	SAS Drug Development Domain	

Program name

It is a common practice in programming to call in several datasets and to use two or more routine macros. Regardless of how many datasets and how many macros used in the program, all of this information is part of the audit process and needs to be documented. In the SAS Drug Development environment, dataset names and macros called within the current program are readily available in the SDDParm dataset. However, by just grabbing all this information and stacking it, will make the audit list messy and cumbersome. To make it more concise and organized, the capability of SEPARATED BY in PROC SQL has the ability to compile the values requested into a list contained within a single macro variable, illustrated as follows:

```
select value into :mtdids_ separated by ', '  from &SDDPARMS
  where ((id="SDTM" or id="INC" or id="CDASH" or id="ADAM" or id="LIB2")
  and (index(value,'.sas7bdat')))
  order by value;

select value into :mtdmcals_ separated by ', ' from &SDDPARMS
  where (index(value,'.sas') and id ne "SETUP" and index(value,'.sas7bdat')=0
  and id ne "<process>" and id ne "MNEW")
  order by value;
```

Parameter ID	Parameter Label	Parameter Type	Value Type	Value
TBLSTAT	Input file	INFILE	filename	tblstat.sas
TBLSTAT	Input file	INFILE	path	/MDS/development/users/ECR/kc9/trial/pksas/...
TBLSTAT	Input file	INFILE	version	<version not available>
TBLSTAT	Input file	INFILE	system	SAS Drug Development Domain
TRT1	Text field	TEXT	value	
TID1	Text field	TEXT	value	
CONCRPT2	Input file	INFILE	filename	concrpt1.sas
CONCRPT2	Input file	INFILE	path	/MDS/development/users/ECR/kc9/trial/pksas/...
CONCRPT2	Input file	INFILE	version	<version not available>
CONCRPT2	Input file	INFILE	system	SAS Drug Development Domain
PARM	Text field	TEXT	value	
M	Text field	TEXT	value	
SIG	Input file	INFILE	filename	sig-conc.sas
SIG	Input file	INFILE	path	/MDS/development/users/ECR/kc9/trial/pksas/...
SIG	Input file	INFILE	version	<version not available>
SIG	Input file	INFILE	system	SAS Drug Development Domain
LIB2	Folder	FOLDER	filename	adpc.sas7bdat
STUDY	Text field	TEXT	value	

Macros called in the program

Dataset used

Using SEPARATED BY in Proc SQL:

idata	macros
adpc.sas7bdat	concrpt.sas, concrpt1.sas, foojust.sas, header.sas, protocol.sas, samptime.sas, setupk.sas, sig-conc.sas, ...

PROC SQL and BASE SAS tools can be used hand in hand to assign macro variables. With this feature of PROC SQL, values assigned in macro variables are easily transformed depending on the desired format as shown below: In SDD, the rundate and runtime follow the ISO 8601 format. For clarity purposes, the line of code below reformats the presentation of the date and time:

```
select put(input(scan(value,1,'T'),yymmdd10.),date9.)||" "||scan(value,-1,'T')
into :mtdts from &SDDPARMS
where upcase(valtype)="DATE";
```

Unfortunately, table and figure titles are not available in the SDDParms dataset. However, given the way our programs are designed, DICTIONARY.TITLES can be used to access table and figure titles presented in the study report. Sometimes the length of the titles will exceed the allowable report margin, and declaring two or more title statements is the only way around this problem. Using SEPERATED BY '~'allows combining several lines of titles into a more readable format, and it can be done using the line of code below.

```
select text into: mtdtit separated by '~' from dictionary.titles
where(type="T");
```

#	Title Locat...	Title Num...	Title Text
1	T	1	Table 14.2.1.1. <Study Drug> Concentrations (ng/mL) by Nominal Time Following
2	T	2	<Description>, (Treatment A)

Using Separated by '~'in Proc SQL:

title
Table 14.2.1.1. <Study Drug> Concentrations (ng/mL) by Nominal Time Following~<Description>, (Treatment A)
Table 14.2.1.2. <Study Drug> Concentrations (ng/mL) by Nominal Time Following~<Description>, (Treatment B)

Once information is collected from individual programs, the second objective of the macro is to collate the metadata by using the assigned macro variables. The macro variables that have been declared become handy in creating the SAS dataset that will subsequently be used to create the audit program list. The advantage of using the macro is that the code is a little less complicated. Using a simple datastep process or PROC SQL CREATE and INSERT statement, the dataset is being populated and updated, as shown below.

```
proc sql;
   create table work.metatemp
      (runtime  Char(30),
       progname Char(40),
       idata    Char(200),
       macros   Char(200),
       title    Char(200));
 insert into work.metatemp
    set runtime="&mtdts.",
        progname="&mtdpnam.",
        idata="&mtdids_.",
        macros="&mtdmcals_.",
        title="&mtdtit.";
quit;
```

```
data work.metadata_injob;
%if %sysfunc(exist(work.metadata_injob) %then
%do;
update work.metadata_injob work.metatemp;
%end;
%else %do;
   set work.metatemp;
%end;
   by title;
run;
data mnew.metadata1;
%if %sysfunc(exist(mnew.metadata1)) %then %do;
update mnew.metadata1 work.metadata_injob;
%end;
%else %do;
set  work.metadata_injob;
%end;
by title;
run;
```

Having a permanent and temporary datasets being created and updated seems very redundant. The reason for having this in the code is because in SAS Drug Development there are two primary methods by which programs are typically submitted. One method is using the Process Editor where a program is submitted individually while the other method is using the Job Editor where programs are run in sequence. When programs are submitted using a Job Editor, the permanent SAS datasets are only updated after the last process has been executed, therefore the permanent metadata reposiroty will only contain information from the last process. Using both a permanent SAS dataset, and a temporary SAS dataset, with a 'reserved' naming convention, we were able to circumvent this difficulty when using the Job Editor.

Unavoidable difficulties may be encountered with the permanent SAS metadata dataset, when after finalizing the submission requirements, sudden changes are requested. Changes in the table and figure titles are most often requested, and while minor, must be accounted for in our audit program list. The code updates by title, thus changing title will cause the duplication of records – the first record contains the previous title while the second record contains the most current one. To guarantee that the record stored in the permanent dataset will be the most recent and the most accurate, a PROC SQL DELETE statement was inserted before updating the permanent dataset to avoid having duplicate records every time a program runs. This line of code will remove any record in the permanent dataset that matches the criteria specified in the WHERE statement. In this particular example, the condition is: if the running program name matches any record in the old file and if the runtime is not the current date and time, the record will be deleted.

```
%if %sysfunc(exist(mnew.metadata1)) %then %do;
proc sql noprint;
  delete from mnew.metadata1
  where (progname = "&mtdpnam.") and runtime ne ("&mtdts.");
quit;
run;
%end;
```

Log:

```
MPRINT(METADATA):   delete from mnew.metadata1 where (progname = "pc-pk-conc-   tables-
sig.sas            ") and runtime ne ("26JAN2011 16:53:50             ");

NOTE: 4 rows were deleted from MNEW.METADATA1.
```

A few examples of requested changes that could affect all of the submitted programs are renumbering and rearranging the tables and figures, altering the format of the headings, or rewording of titles. Typically, a programmer or statistician will use the job editor to generate the desired outputs and together with this, rebuild the metadata of program settings. Keeping in mind

that the job editor only updates the permanent dataset after the execution of the last program in the list, it is required then to keep track of all the previous program names to make sure that after the submission of the job, all old records of the previous run programs are also been removed. If there will be no place holder for the other submitted programs, the only record that will be updated is the one that is related to the last program in the list. To address this problem, the code above was modified. The code uses SELECT DISTINCT INTO which is used to pass the unique values of a variable and be declared as a macro variable. By using the QUOTE function, every value that is passed into a macro variable will be in double quotation so that it will be treated as a character when called.

```
%if %sysfunc(exist(work.metadata_injob)) %then %do;
proc sql noprint;
   select distinct quote(progname) into: prg separated by ',' from work.metadata_injob;
quit;
run;
%let prg=&prg;
%end;

%if %sysfunc(exist(mnew.metadata1)) %then %do;
proc sql noprint;
  delete from mnew.metadata1
  where (progname in (&prg) and runtime ne ("&mtdts."));
quit;
run;
%end;
```

Log:

```
MPRINT (METADATA):   delete from mnew.metadata1 where (progname in ("pc-kel-tables.sas
","pc-pk-conc-tables-sig.sas
            ","pc-pk-stats.sas                          ","pc-pk-tables.sas
") and runtime ne ("26JAN2011 17:19:13         "));

NOTE: 9 rows were deleted from MNEW.METADATA1.
```

The output of the program is a metadata SAS dataset where each row corresponds to the program settings related to an output included in a clinical study report as shown below. Having this dataset makes the process of program documentation faster since it is readily available after running all the programs. If there are necessary adjustments to be made to meet client specification, it can be accessed and retrieved through SAS for further mining.

#	runtime	progname	idata	macros	title
1	14JUL2010 21:18:01	SDTM-LIS-LNO.SAS	adsl.sas7bdat, co.sas7bdat, l...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.1.10.1. Clinical Laboratory Reference Ranges
2	14JUL2010 21:16:03	SDTM-LIS-DIS.SAS	adsl.sas7bdat, co.sas7bdat, ...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.1. Subject Discontinuation
3	14JUL2010 21:15:53	SDTM-LIS-DEM.SAS	adsl.sas7bdat, co.sas7bdat, ...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.4.1. Demographics
4	14JUL2010 21:14:10	CDASH-LIS-PHY-BES...	adsl.sas7bdat, pe.sas7bdat	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.4.2. Physical Examination
5	14JUL2010 21:19:00	SDTM-LIS-MH.SAS	adsl.sas7bdat, mh.sas7bdat,...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.4.3. Medical and Surgical History
6	14JUL2010 21:19:14	SDTM-LIS-SU.SAS	adsl.sas7bdat, su.sas7bdat	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.4.4. Substance Use
7	14JUL2010 21:16:38	SDTM-LIS-INC.SAS	adsl.sas7bdat, ieop.sas7bdat	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.1.1. Inclusion Criteria
8	14JUL2010 21:16:23	SDTM-LIS-EXC.SAS	adsl.sas7bdat, ieop.sas7bdat	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.1.2. Exclusion Criteria
9	14JUL2010 21:16:31	SDTM-LIS-IE.SAS	adsl.sas7bdat, ie.sas7bdat, i...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.2. Subject Eligibility
10	14JUL2010 21:15:21	SDTM-LIS-CHK.SAS	adsl.sas7bdat, mqop.sas7bd...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.3.1. Check-in and Return Criteria
11	14JUL2010 21:15:28	SDTM-LIS-CHK1.SAS	adsl.sas7bdat, mqop.sas7bd...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.3.2. Check-in and Return Responses
12	14JUL2010 21:18:24	SDTM-LIS-MED.SAS	adsl.sas7bdat, ex.sas7bdat, ...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.4.1. Test Compound Description
13	14JUL2010 21:18:37	SDTM-LIS-MED2.SAS	adsl.sas7bdat, co.sas7bdat, ...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.4.2. Test Compound Administration Times
14	14JUL2010 21:15:13	SDTM-LIS-BLD.SAS	adsl.sas7bdat, pcop.sas7bdat	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.5. Blood Draw Times
15	14JUL2010 21:18:51	SDTM-LIS-MEL.SAS	adsl.sas7bdat, ml.sas7bdat, ...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.6. Meal Times
16	14JUL2010 21:15:41	SDTM-LIS-CON.SAS	adsl.sas7bdat, cm.sas7bdat,...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.5.7. Concomitant Medications
17	30JUL2010 14:21:03	SDTM-LIS-AE.SAS	adsl.sas7bdat, ae.sas7bdat, ...	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.7.1. Adverse Events (I of II)
18	14 JUL 2010 21:14:44	SDTM-LIS-AE2.SAS	adsl.sas7bdat, ae.sas7bdat	CDATETIME.sas, MERGE_SUPP.sas, ...	Appendix 16.2.7.2. Adverse Events (II of II)

Once this metadata SAS dataset has been created, it can easily be transformed into an Excel file, or alternative format, for delivery to a client. Moreover, the output can effectively be used by non-SAS users for different tasks such as cross-referencing programs with the output, verifying titles, or generating table of contents.

CONCLUSION

This paper has illustrated a method of tracking submission metadata within the SAS Drug Development environment, specifically using the SDDPARMS dataset. Using PROC SQL and the SAS DATA step, we were able to write a non-invasive macro that efficiently collates all information necessary to build an audit program list. The code presented can be easily modified if there are additional items required in the audit program list by merely adding additional sub-queries to the SQL procedure.

REFERENCES

SAS 9.1.3 Online Documentation

SAS Drug Development 3.4 User's Guides

Shu, Amos. *The SAS Programming Experience in the SAS Drug Development (SAS DD) Environment – Comparing with the Regular SAS Programming Environment.* Proceedings of NESUG 2010.

ACKNOWLEDGMENTS

We would like to thank our colleagues in the Biostatistics and Pharmacokinetics groups in Clinical Pharmacological Sciences department at Celerion who provided comments on earlier drafts. Their insights and guidance are appreciated.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact the authors at:

Katrina E. Canonizado
Celerion
621 Rose Street
Lincoln, NE 68502
katrina.canonizado@celerion.com

Bradford J. Danner
i3 Statprobe
brad.danner@i3statprobe.com

Matthew J. Wiedel
Celerion
621 Rose Street
Lincoln, NE 68502
matthew.wiedel@celerion.com

APPENDIX

MACRO CODE

```
%let choice = YES;

libname mnew "&mnew";

%macro metadata;

%if &choice=YES %then %do;

%global mtdpnam mtdids_ mtdmcals_ mtdtit mtdts;

proc sql noprint;
select value into :mtdpnam from &SDDPARMS where (id="<process>" and valtype="filename");
select value into :mtdids_ separated by ', '  from &SDDPARMS
      where ((id="SDTM" or id="INC" or id="CDASH" or id="ADAM" or id="LIB2") and
(index(value,'.sas7bdat'))) order by value;
select value into :mtdmcals_ separated by ', ' from &SDDPARMS
where (index(value,'.sas') and id ne "SETUP" and index(value,'.sas7bdat')=0 and id ne
"<process>" and id ne "MNEW") order by value;
select text into: mtdtit separated by '~' from dictionary.titles where(type="T");
select put(input(scan(value,1,'T'),yymmdd10.),date9.)||" "||scan(value,-1,'T') into :mtdts
from &SDDPARMS where upcase(valtype)="DATE";
quit;

proc sql noprint;
   create table work.metatemp
      (runtime  Char(30),
       progname Char(40),
       idata    Char(200),
       macros   Char(200),
       title    Char(200));

 insert into work.metatemp
    set runtime="&mtdts.",
        progname="&mtdpnam.",
        idata="&mtdids_.",
        macros="&mtdmcals_.",
        title="&mtdtit.";
quit;

data work.metadata_injob;
%if %sysfunc(exist(work.metadata_injob)) %then %do;
   update work.metadata_injob work.metatemp;
%end;
%else %do;
   set work.metatemp;
%end;
   by title;
run;

%if %sysfunc(exist(work.metadata_injob)) %then %do;
proc sql noprint;
   select distinct quote(progname) into: prg separated by ',' from work.metadata_injob;
quit;
run;
%let prg=&prg;
%end;

%if %sysfunc(exist(mnew.metadata1)) %then %do;
proc sql noprint;
  delete from mnew.metadata1
  where (progname in (&prg.) and runtime ne ("&mtdts."));
quit;
run;
```

```
%end;

data mnew.metadata1;
%if %sysfunc(exist(mnew.metadata1)) %then %do;
update mnew.metadata1 work.metadata_injob;
%end;
%else %do;
set work.metadata_injob;
%end;
by title;
run;
%end;

%mend metadata;
```