PharmaSUG2013 – Paper AD11

Let SAS Set Up and Track Your Project

Tom Santopoli, Octagon, now part of Accenture
Wayne Zhong, Octagon, now part of Accenture

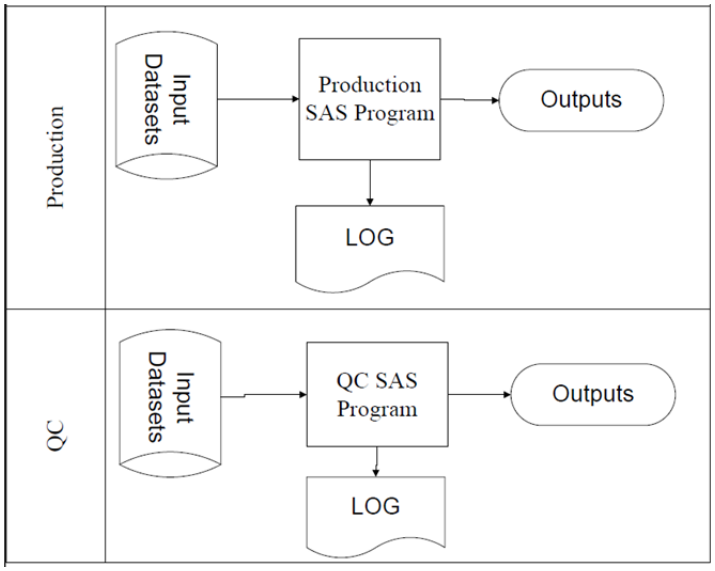
ABSTRACT

When managing the programming activities in a clinical trial, a project tracking document can be a convenient place to store important information. This information may include an inventory of the datasets, tables, listings, and figures (DTLFs), the programs that produce the DTLFs, and the validation/completion status of the DTLFs. SAS can use this information to dynamically generate program shells and batch scripts before any of the programming work even begins. In addition, SAS can update the validation/completion status of each DTLF by examining the properties and contents of logs and other files in the project folder. In this paper, we will present several very powerful techniques that let SAS do all of this work for us, thereby ensuring high quality completion of the programming tasks in a clinical trial.

INTRODUCTION

In the pharmaceutical environment, a common project for SAS programmers is the creation of FDA submission deliverables. This project takes SAS datasets as raw material and produces outputs including analysis SAS datasets, tables, listings, and figures. Two types of tasks comprise the majority of work for this project: 1) writing SAS production programs that take inputs and produce outputs 2) writing SAS QC programs that use the same inputs to validate the production outputs. The act of running a SAS program also generates a log describing the state of code execution. Together, the input, output, and log – as displayed in Diagram 1 – describe the status of each production and QC task.

Diagram 1



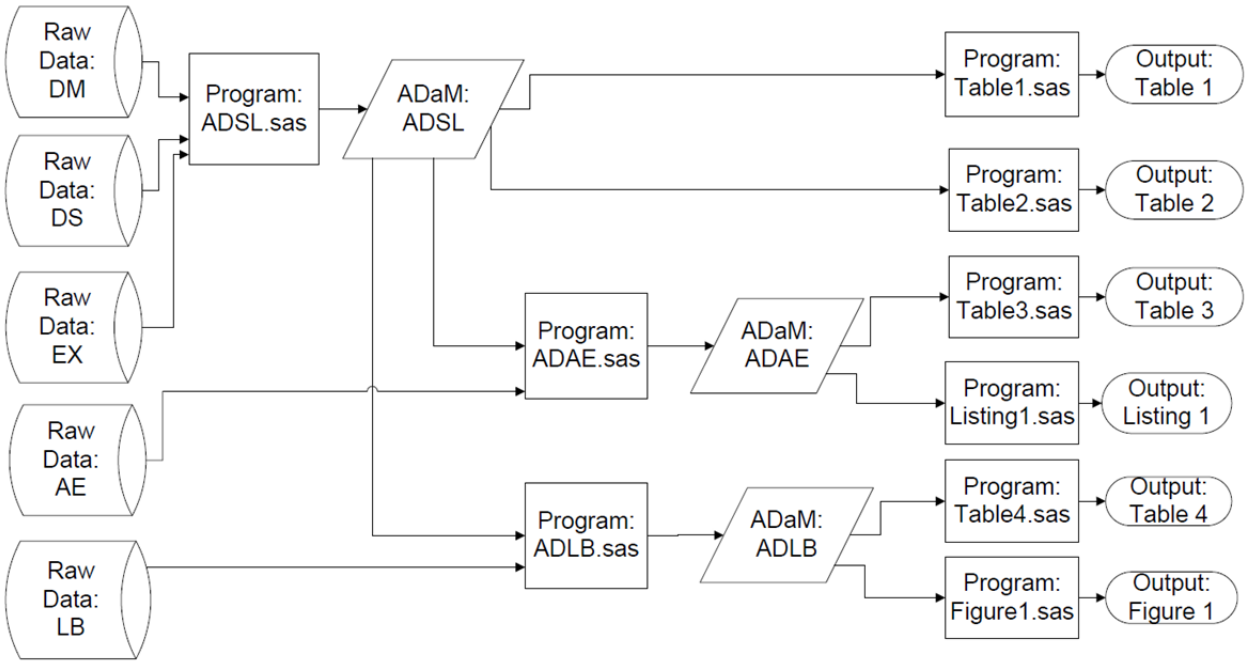
An output is considered ready for final visual inspection when both the production and QC programs are complete, the values agree, and the respective logs are free of issues. However, should any input data be updated, all the steps must be repeated again to ensure the data is analyzed correctly.

Time constraints usually mean that work starts with draft raw data, which will lead to draft analysis data, and finally draft TLF reports. As raw data goes through various reviews and updates, invariably all the programs downstream must be updated repeatedly, putting challenges on determining the status of tasks.

PROJECT TRACKING

In one project, there could be dozens to hundreds of tasks, creating challenges due to both volume and dependencies between tasks. Displayed in Diagram 2 below is a sample flow of nine outputs (3 analysis datasets and 6 TLFs), and their relationship to each other and to the raw data. For example, if DM is updated all programs are impacted, whereas only 3 programs are impacted if LB is updated. Please note that only the production side of each task is shown below. In actuality there would be an additional 9 QC programs for a total of 18 programs.

Diagram 2



Reviewing and recording the status of tasks is typically left to individual programmers for the majority of the project. To determine that one task is complete, one would need to do the following:

1. Visually inspect the production output
2. Compare the output and QC results
3. Ensure the latest data is used
4. Verify a log is present and has no issues
5. Verify that relative time stamps are appropriate:
    a. Output and log dates are equal
    b. All outputs/QC results newer than last program modified time
    c. QC occurred after output creation

The time needed to review even one task is not trivial. Additionally, this would need to be repeated with new raw data cuts and program version updates. As a result, getting an accurate status report for the entire project at any point in time is difficult due to both the frequency of updates and the amount of checks needed.

The only time when a comprehensive check can take place is prior to a delivery, which is when all tasks should be completed. At that point all programs can be rerun and rechecked en masse. However, as this is the most hectic time during a project, issues discovered may be difficult to correct, or perhaps even worse, may be missed entirely.

THE SOLUTION

Every project already uses a tracking document as a central location to list tasks and to track the status of each task. The problem is that maintaining this document is often an entirely manual process. SAS has the capability to retrieve the names, dates, and contents of SAS programs, logs, and DTLF files within the project folder. With the exception of visual inspection of the production outputs, SAS can then automatically perform the checks specified above and update the tracking document dynamically.

BASIC TRACKING DOCUMENT

Table 1

Type	Program Name	Output	Title	Prod. Author (Last Modified)	Prod. Status	Prod. Comment	Valid. Author (Last Modified)	Valid. Status	Valid. Comment
Dataset	ADSL.sas	ADSL	Subject Level Analysis Dataset	Wayne (2013-03-14)	75%		Jaya (2013-03-14)	75%	
Dataset	ADAE.sas	ADAE	Adverse Events Analysis Dataset	Wayne (2013-03-14)	75%		Ravi (2013-03-15)	50%	QC result not passed
Dataset	ADLB.sas	ADLB	Lab Analysis Dataset	Wayne (2013-03-14)	50%	Newer input data available	Jaya (2013-03-14)	50%	Production not at 75%
Table	TABLE1.sas	Table 1	Summary of Disposition	Tom (2013-03-12)	50%	Log check not passed	Jaya (2013-03-13)	50%	Production not at 75%
Table	TABLE2.sas	Table 2	Summary of Demographics	Tom (2013-03-12)	75%		Jaya (2013-03-13)	50%	QC result older than output
Table	TABLE3.sas	Table 3	Overview of TEAEs	Tom (2013-03-12)	75%		Jaya (2013-03-13)		
Table	TABLE4.sas	Table 4	Summary of Lab Results	Tom (2013-03-12)	50%	Newer input data available	Jaya (2013-03-13)	50%	Production not at 75%
Listing	LISTING1.sas	Listing 1	Listing of AEs	Tom (2013-03-12)	75%		Latha (2013-03-14)	25%	
Figure	FIGURE1.sas	Figure 1	Figure of Hy's Law	Tom (2013-03-12)	50%	Newer input data available	Latha (2013-03-14)	50%	Production not at 75%

In the sample tracking document above, the first three columns give the task type, SAS program name, and output name. These three identifiers can be used to determine the files that should exist for each task, and to instruct SAS to locate and analyze them. For example, the first task above represents the ADSL dataset. The following files are associated with this task:

1. ADSL.sas – the production program
2. ADSL.log – the production log
3. ADSL.sas7bdat – the production output
4. QC_ADSL.sas – the QC program
5. QC_ADSL.log – the QC log

The list of files across all tasks represents the project organization. SAS can automatically perform the checks in the Project Tracking section of this paper by examining the date stamp of these files and the contents of the log files. The check results are then dynamically updated in the last six columns. The percentages and colors shown are arbitrarily set depending on checks passed. Please note that the one check SAS cannot perform is a visual inspection of the production output.

The organizational structure defined by the tracking document above also allows for the implementation of many useful project utilities. For example, program shells containing the program header and calls to standardized macros can be dynamically added to the project folder for new tasks. Batch scripts can be generated for specified subsets of tasks. Titles and footnotes for TLF outputs can be linked to each task. Task outputs can be selectively moved to a delivery folder based on their completion status. The generation of program shells and batch scripts from the tracking document are among the techniques presented in the remaining sections of this paper.

SAS TECHNIQUES

Retrieve a list of files from a specified folder:

```
1   filename inpath pipe ' dir "C:\your_path\" ';

2   data meta;
3       length line $2000;
4       infile inpath length=lenvar;
5       input @1 line $varying. lenvar;
6   run;
```

In line 1, the fileref **inpath** with **pipe** option allows SAS to read in a list of files from the specified path.
In lines 2-6, the infile statement reads the contents of the fileref **inpath** into SAS dataset META. The content of META is displayed in Table 2.

Table 2:
META

04/12/2013	04:23 PM	6,441 AD2CM.sas
04/12/2013	06:46 PM	15,038 AD2EG.sas
04/10/2013	04:11 PM	23,123 AD2EX.sas
04/12/2013	01:03 PM	22,313 AD2LB.sas
04/12/2013	07:56 PM	8,949 AD2QS.sas

SAS functions can be used to extract the file name, file extension, date time stamp, and file size from META. This information can be used for a variety of purposes, such as: 1) writing the date stamp of a program's input datasets into the log file 2) comparing program, DTLF, and log date stamps 3) checking whether files defined in the tracking document are present in the project folder.

Programmatically perform a log check:

```
1   filename inlog "C:\your_path\your_log.log";

2   data log;
3       length line $1000;
4       infile inlog length=lenvar;
5       input @1 line $varying. lenvar ;
6       if lowcase(scan(line,1,' ,:()')) in
7             ('note', 'error', 'warning','info');
8   run;
```

In line 1, the fileref **inlog** specifies the log file to be checked.
In lines 2-8, the infile statement reads the contents of the log file into the SAS dataset LOG. The content of LOG is displayed in Table 3.
In line 6, relevant messages in the log are kept to be analyzed.

Table 3:
LOG

NOTE: PROCEDURE SQL used (Total process time):
NOTE: There were 3308 observations read from the data set WORK.BASE3.
NOTE: The data set WORK.VS2 has 1654 observations and 4 variables.
NOTE: PROCEDURE TRANSPOSE used (Total process time):

SAS functions can be used to identify messages that are considered unacceptable. Other useful information can be parsed out from log messages, such as: 1) the date time stamp of datasets intentionally written to the log 2) the results of QC checks intentionally written to the log.

Writing a PROC COMPARE result to the log:

```
1   proc compare base=your_base compare=your_comp;
2   run;

3   data _null_;
4       if &sysinfo>0 then put 'NOTE: RESULT=FAIL';
5       else put 'NOTE: RESULT=PASS';
6   run;
```

In lines 1-2, a normal PROC COMPARE is used. The automatic macro variable SYSINFO stores a value (an integer >=0) that represents the result of the comparison. A value of 0 indicates the comparison is perfect, and higher values indicate increasing differences.
In lines 3-6, the value of SYSINFO is checked and the result is written to the log.

Generating Program Shells:

A SAS program can incorporate multiple global macros designed to set library references, enable automated status tracking, and other functions needed. The use of a program shell ensures that all required code is present in every program. The following is an example of a basic program shell:

```
%lstlog;
/*************************************************************************
* Program Name       :
* Programmer         :
* Date Created       :
* Description        :
*************************************************************************/

%include "&progloc\..\Maclib\msetup.sas";
%cleanup;
%create (datalist=xxx xxx, inlib=sdtmlib) ;

**---------------- WRITE PROGRAM CODE HERE  -------------------------**;

**---------------- END PROGRAM CODE         -------------------------**;
**  QC RESULT  **;
%compare (base=xxx, compare=xxx);

**  LOG CHECK  **;
%endprog;
```

In the program shell above, the macro ***%create*** is used to create work versions of permanent datasets that serve as input to the program. The macro also records the version of datasets used by writing the date stamp of each dataset to the log. Similarly, ***%compare*** records the result of a QC check in the log.

Two lists are necessary to generate program shells: 1) a list of all programs as determined using the tracking document, obtained by importing the tracking file 2) a list of programs that currently exist in the project folder, obtained by using the pipe code above. Only items in 1 and not in 2 are used to generate program shells. This avoids overwriting programs that already exist. The code below generates the program shell:

```
1   data shell;
2       infile "C:\in_path\shell.sas" length=lenvar;
3       input @1 name $varying. lenvar;
4   run;
```

In lines 1-4, the infile statement is used to read the program shell into SAS dataset SHELL.

SAS functions can then be used to dynamically update header information, values of parameters in macro calls, etc. The following code outputs the updated contents of the dataset SHELL to the designated file.

```
5   data _null_;
6       set shell;
7       file "C:\out_path\ADSL.sas ";
8       len=length(name);
9       put name $varying. len;
10  run;
```

Generating Batch Scripts:

Before the following code, the dataset PROGLIST, which contains a list of programs in both the tracking document and the project folder, is created. The process to do so is explained in the section Generating Program Shells.

```
1   filename batchfile "your_path\batch.bat";

2   data _null_;
3       set proglist;
4       run='sas ' || strip(progname);
5       file batchfile;
6       if _n_=1 then put 'path=%PATH%;c:\SAS\SASFoundation\9.2';
7       put run;
8   run;
```

In line 1, the fileref **batchfile** specifies the batch file to be created.
In line 4, the variable RUN contains the concatenation of the string ‘sas’ with the variable PROGNAME, which contains the SAS program name.
In line 6, the string specified is written as the first line of the batch file, and points to the location of the SAS.exe file in the computing environment. The location will be unique for each environment.
In line 7, the values of the variable RUN are written to the batch file.

```
path=%PATH%;c:\SAS\SASFoundation\9.2
sas ADSL.sas
sas ADAE.sas
sas ADLB.sas
```

An example of a batch script is shown above. Please note that this batch script will only function if it resides at the same path as the SAS programs it will execute. One benefit of this approach to creating batch scripts is that only programs of a certain completion status can be selected for the script.

Updating an Excel Spreadsheet with Dynamic Data Exchange (DDE):

Before the following code, the dataset NEWSTATUS, which contains the data used to populate columns 5 through 10 in Table 1, is created. The theory and techniques discussed in the other sections of this paper are applied to create this dataset.

```
1   options noxsync;

2   x ' "c:\your_path\track.xls" ';

3   filename update dde "excel|status!r1c6:r30c6" notab;

4   data _null_;
5       set newstatus;
6       file update;
7       put percent;
8   run;

9   filename system dde "excel|system";

10  data _null_;
11      file system;
12      put '[error(false)]';
13      put '[save.as("c:\your_path\track.xls")]';
14      put '[close("c:\your_path\track.xls")]';
15  run;
```

In line 1, the option NOXSYNC allows the code in line 2 to run without user input.
In line 2, the tracking document is opened.
In line 3, the fileref **update** with the **dde** option specifies rows 1 through 30 in column 6 of the worksheet “status” within track.xls.
In lines 4-8, the content of variable PERCENT in dataset NEWSTATUS is written to the cells specified by fileref **update**.
In line 9, the fileref **system** is created to send system commands to excel.
In lines 10-15, commands to save and close the track.xls file are sent to excel.

CONCLUSION

Collectively, the techniques presented in this paper establish the project tracking document as the centralized location to obtain accurate, reliable information regarding the status of the project. An up-to-date tracking document allows programmers to become more effective by laying out tasks and issues clearly. Automating many status checks with SAS saves time and eliminates human error. All of this contributes to a smoother and higher quality project.

Beyond benefits to the immediate project, the data collected by SAS can feed into project reports summarizing tasks completed and resources used. This can also enable evaluations of performance over time as well as the impact of process changes.

ACKNOWLEDGMENTS

We would like to thank Hal Li for his help with putting together this presentation. We would also like to thank the entire clinical programming department at the Octagon unit of Accenture, who continue to help us refine this process through their valuable feedback.

CONTACT INFORMATION

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

Author Name:	Tom Santopoli
Company:	Octagon, now part of Accenture
Address:	585 East Swedesford Road, Wayne, PA
Work Phone:	(610) 535-6500 x5613
Email:	thomas.r.santopoli@accenture.com
Web:	www.octagonresearch.com

Author Name:	Wayne Zhong
Company:	Octagon, now part of Accenture
Address:	585 East Swedesford Road, Wayne, PA
Work Phone:	(610) 535-6500 x5535
Email:	wayne.zhong@accenture.com
Web:	www.octagonresearch.com