PharmaSUG 2012 – AD19

Making a List, Checking it Twice: Efficient Production and Verification of Tables and Figures Using SAS®

Linda Collins, PharmaStat, LLC, Newark, CA

Elizabeth Li, PharmaStat, LLC, Newark, CA

ABSTRACT

Managing all the 'moving parts' of clinical data analysis is a daunting task. 'Moving parts' include inputs (raw data files, specifications), outputs (analysis files and reports) and processes (programs and macros). Each process and output must be verified as correct and each program executed in the proper sequence. Project leaders must be able to track the verification status of each output. In this paper, we present techniques for managing analysis through an 'inventory' spreadsheet of analysis metadata. The metadata is used to generate 'driver' programs that perform data selection and produce the tables, listings and figures (TLFs). Analysis programs not only generate TLFs, but also create SAS® datasets with a consistent format for storing summary results. The results datasets make it possible to automate independent programming verification. In addition, it makes possible to track the status of the entire project in a summary report by using the information from both the metadata and the results datasets. Furthermore, the inventory approach is expected to be highly useful when standards for analysis results metadata in define.xml are published.

KEY WORDS

Automated programming, SAS® program verification, program specifications, independent programming, analysis results metadata

INTRODUCTION

Managing the production and quality of the SAS® outputs, such as tables, listings and figures (TLFs), can be an overwhelming task in FDA regulated industries like biotechnology and pharmaceutical. In this paper, we present techniques to improve programming efficiency of data analysis and control quality of analysis results. This approach starts with recording analysis results metadata in a TLF 'inventory' spreadsheet. From the metadata we automatically generate a 'driver' program that performs data selection for each TLF. By using SAS® macros, we further automate the verification of the analysis results through independent programming. At the end of this process, a final status report can be generated to document the quality status of all tables and figures. These techniques can be summarized as the following steps:

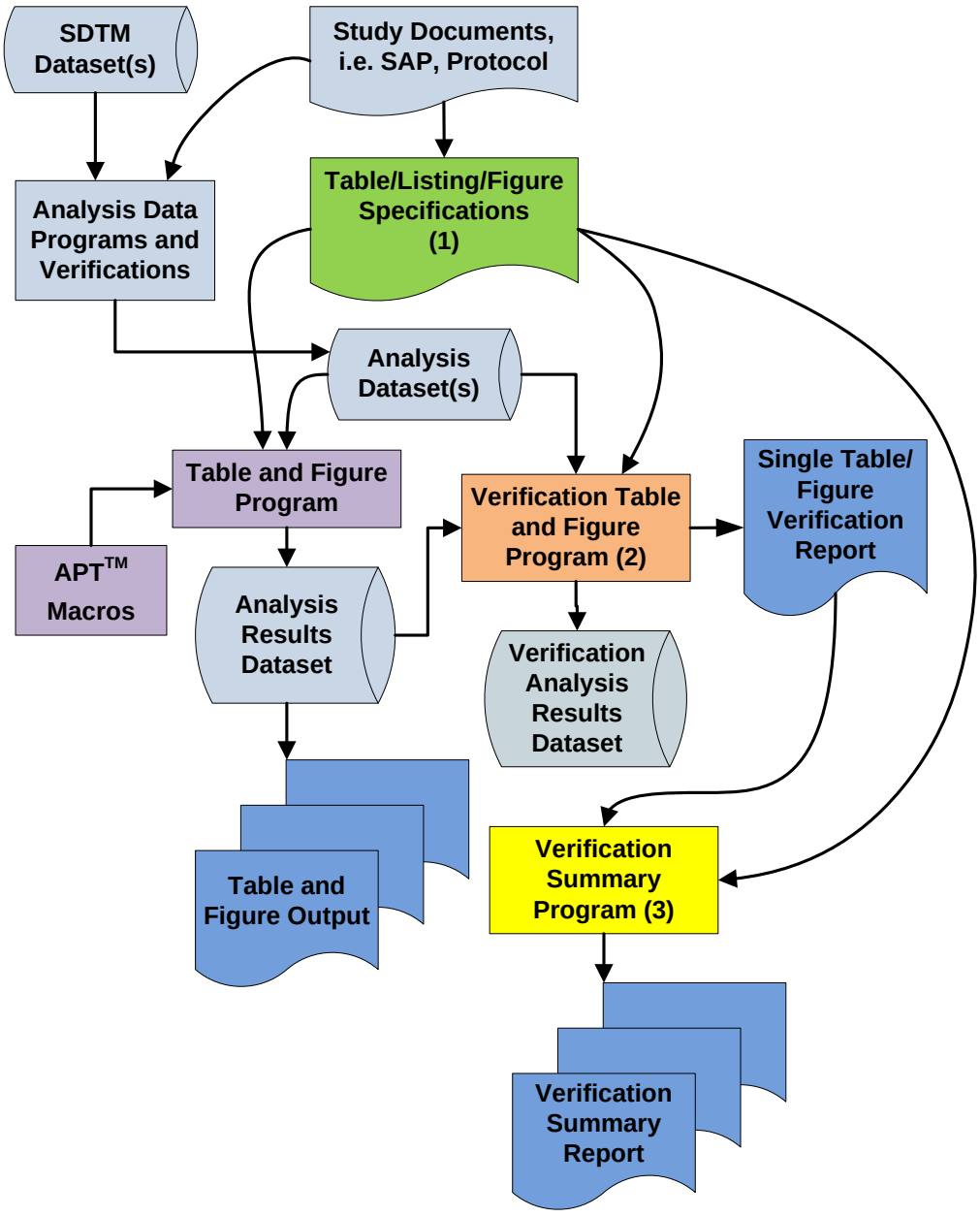
(1) Documentation of specifications

(2) Independent verification

(3) Automated quality status report

Figure 1 is a flow chart that depicts the process, where the corresponding steps (1), (2), and (3) are indicated. Following sections present details of the techniques.

Figure 1. Tables and Figures Production Process Flow Chart



MAKING A LIST

Specifications are the key to efficient programming. We all know "there's never enough time to do it right but there's plenty of time to do it over". Programming efficiency is achieved by "doing it right the first time". Clear programming specifications are the foundation. The only way to do it right is to know what 'right' means. Programming specifications include, but are not limited to, the following:

1) What type of tables and figures are produced?
2) How are they connected to the mock-up shells?
3) What input datasets and variables are used?
4) What subsets of data are included?
5) What statistics will be generated?
6) What statistical tests, p-values will be reported for treatment group differences?
7) What are the titles and footnotes of each TLF?
8) SAS® program name for each TLF

In addition, management of quality control (QC) process includes, but not limited to, the following:

1) What are the discrepancies in the results between production and verification for each table/figure?
2) What is the verification status of each table/figure?
3) What is the percentage of total outputs that have been verified?
4) What is the visual review status of each TLF?

A statistical analysis plan (SAP) with mock-up shells of TLFs is the preferred starting point for the specifications. If such a document is not available, study protocol may provide a general idea of what type of data analyses or summaries are required. Mock-up shells are great visual aid to project what the output will look like. Here is an example of annotated mock-up table, which is a tool to identify the analysis data, analysis variables, and any subset conditions (marked in red text) to be used in data analysis. This example was taken from a re-analysis of a U.S Public Health Service study of rifapentine and isoniazid in patients with HIV-related tuberculosis [1].

[1] Centers for Disease Control, U.S. Public Health Service Study 22, conducted by the Tuberculosis Trials Consortium. Used with permission. Study design is available at http://clinicaltrials.gov/ct2/show/record/NCT00023335

Figure 1. Sample Annotated Table Mock-up Shell

adamdata.ADSL
usubjid trt01pn
hivpfl complfl
st04mofn
st24mofn

seg	linlabel	col1	col2	col3		Notes
	Source 1 Table 5. Clinical Outcome in HIV Negative Patients with Pulmonary Tuberculosis					All: hivpfl='N'
		Rifapentine Combination Treatment n/N (%)	Rifampin Combination Treatment n/N (%)	p-value	usubjid trt01pn hivpfl complfl	
1	**Status at End of 4 Months Continuation Phase**					
	Treatment Response[a]	xx/xxx (xx.x%)	xx/xxx (xx.x%)	0.xxxx	st04mofn	P-value is chi-square comparing treatment groups.
	Not Converted	xx/xxx (xx.x%)	xx/xxx (xx.x%)			
	Did Not Complete Treatment[b]	xx/xxx (xx.x%)	xx/xxx (xx.x%)			
	Death[c]	xx/xxx (xx.x%)	xx/xxx (xx.x%)			
2	**Status Through 24 Month Follow-up**				st24mofn	Only this segment: complfl='Y'
	Relapsed	xx/xxx (xx.x%)	xx/xxx (xx.x%)	0.xxxx		P-value is chi-square comparing treatment groups.
	Sputum Negative	xx/xxx (xx.x%)	xx/xxx (xx.x%)			
	Lost to Follow-up	xx/xxx (xx.x%)	xx/xxx (xx.x%)			
	Deaths	xx/xxx (xx.x%)	xx/xxx (xx.x%)			

Based on the study information, we can specify the TLFs to be produced in an Excel file (we refer to this as the electronic table of contents, or eTOC.xls). The advantage of using an Excel file is that it is machine readable. It can be used as a source for a SAS® macro program to generate a "driver" program for generating outputs, as well as for automated program of quality status report. This specification document is a central reference for analysis methods, in addition to SAP. It usually provides more programming details than SAP for how outputs are generated.

Each report is assigned a 'base program' name and a suffix. The 'base program' name corresponds to the name of a report macro with the same name, stored in the local macros folder. A 'driver generator' program reads the eTOC file and writes a short 'driver' program for each report output. 'Driver' programs contain a section of code to access the input data according to the specifications, and then a call to the report macro.

Figure 2 is an example of programming specifications (eTOC) in Excel structure.

Figure 2. Sample Excel File That Stores Specifications

ReportNo	Titles	BaseProg	Order	Variant
Source 1 Table 5	Clinical Outcome in HIV Negative Patients with Pulmonary Tuberculosis	ts1t5	hivn	01
Source 4 Table 3	Clinical Outcome in HIV Positive Patients with Pulmonary Tuberculosis	ts1t5	hivp	02

Figure 2. Sample Excel File That Stores Specifications (continued)

FNRef	Dataset1	Select1	Vars1	Specifications	QC Level	QC Visual	QC Stat
F1, F2, F3, F4	adamdata. ADSL	HIVPFL= 'N'	usubjid trt01pn hivpfl complfl st04mofn st24mofn	Segment 1: include all pts with HIVPFL='N'. Chi-Square is used to compare trt01pn. Segment 2: include all pts with HIVPFL='N' and complfl='Y'. Chi-Square is used to compare trt01pn.	2	Compl eted	Done
F1, F2, F3, F5	adamdata. ADSL	HIVPFL= 'Y'	usubjid trt01pn hivpfl complfl st04mofn st24mofn	Segment 1: include all pts with HIVPFL='Y'. Fisher's exact test is used to compare trt01pn. Segment 2: include all pts with HIVPFL='Y' and complfl='Y'. Fisher's exact test is used to compare trt01pn.	2	On-going	Done

A driver SAS® program can be created based on the information in Figure 2. Here is the generated code:

```
/** ------------------------------------------------------------------------------**
** Program:    ts1t5-hivn.sas          BaseProg -Order
** Order:      hivn                    Order
** Variant:    01                      Variant
** Generated:  07NOV2011
** Report No:  Source 1 Table 5        ReportNo
** Title 1:    Clinical Outcome in HIV Negative Patients with Pulmonary Tuberculosis  Titles1
** Inputfile:  adamdata.ADSL           Dataset1
** Outputfile: ts1t5-hivn-01.rtf       BaseProg –Order-Variant
** ------------------------------------------------------------------------------**/
```

```
data file1 ;                    Dataset1
    set adamdata.ADSL ;
    where (HIVPFL='N') ;        Select1
    keep  usubjid trt01pn hivpfl complfl st04mofn st24mofn;      Vars1
run ;

%ts1t5 (                                BaseProg
                order    = hivn ,
                variant  = 01
               ) ;
```

The code used to produce this 'driver' program is shown in Appendix A at the end of this paper.

Each driver program produces a single output. However, a base program may be used for any number of variations, usually subset analyses. The programmers do not modify the driver programs. Any changes are made in the eTOC Excel file. The driver generator program is rerun subsequently. The programmers write the report macro that is called by the driver program. The report macro is tested using the driver or drivers.

Our company uses an internally developed SAS® macro library, analysis productivity tools (APT™), for clinical data analysis. Using the APT™ macros, a programmer can easily generate summary tables.

Figure 3 is a sample output from APT SAS® macros

Figure 3 Sample Output:
Source 1 Table 5 Clinical Outcome in HIV Negative Patients with Pulmonary Tuberculosis

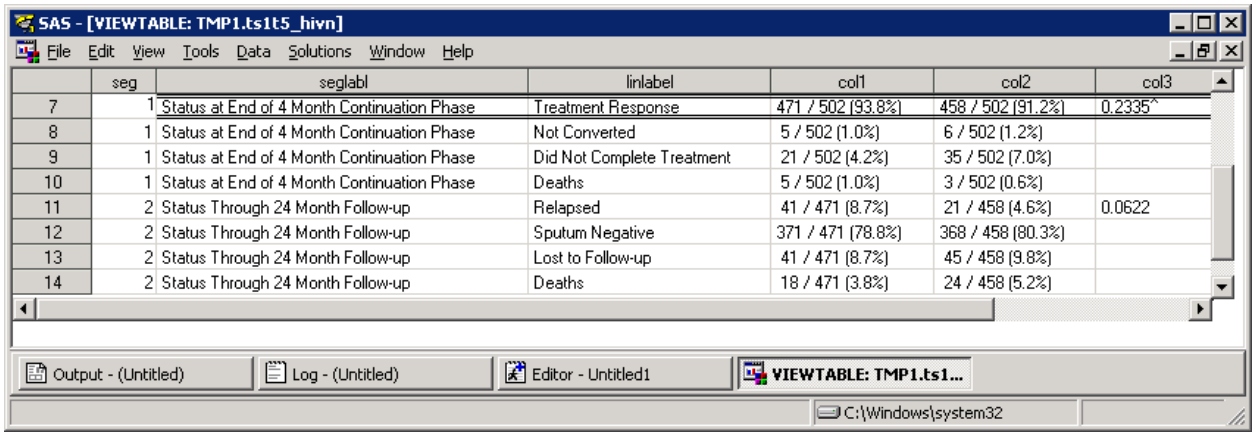
		Once-weekly isoniazid/rifapentine n/N (%)	Twice-weekly isonizid/rifampin n/N (%)	p-value
Status at End of 4 Month Continuation Phase				
	Treatment Response	471 / 502 (93.8%)	458 / 502 (91.2%)	0.2335^
	Not Converted	5 / 502 (1.0%)	6 / 502 (1.2%)	
	Did Not Complete Treatment	21 / 502 (4.2%)	35 / 502 (7.0%)	
	Deaths	5 / 502 (1.0%)	3 / 502 (0.6%)	
Status Through 24 Month Follow-up				
	Relapsed	41 / 471 (8.7%)	21 / 458 (4.6%)	0.0622
	Sputum Negative	371 / 471 (78.8%)	368 / 458 (80.3%)	
	Lost to Follow-up	41 / 471 (8.7%)	45 / 458 (9.8%)	
	Deaths	18 / 471 (3.8%)	24 / 458 (5.2%)	

^ Warning: 25% of the cells have expected counts less than 5. Chi-Square may not be a valid test.
Report Status: DRAFT Created: 18DEC11 09:53 Source: CDC_TB\TLGs\ts1t5_hivn.sas

One essential feature of this process is that the production program generates not only an output file, but also a results SAS® dataset, which has standardized structure. The results from a verification program are stored in a SAS dataset with the same structure. This makes it possible to automate the comparison of the analysis results from production and verification.

The saved results dataset is shown below. The variable SEGLABL identifies the section or 'segment' of statistics and LINLABEL contains the labeling information for a specific row. The statistics are set up as formatted character strings in variables COL1 through COL3.

Figure 4. Saved Results Dataset for Report "Source 1 Table 5 Clinical Outcome in HIV Negative Patients with Pulmonary Tuberculosis"



	seg	seglabl	linlabel	col1	col2	col3
7	1	Status at End of 4 Month Continuation Phase	Treatment Response	471 / 502 (93.8%)	458 / 502 (91.2%)	0.2335^
8	1	Status at End of 4 Month Continuation Phase	Not Converted	5 / 502 (1.0%)	6 / 502 (1.2%)	
9	1	Status at End of 4 Month Continuation Phase	Did Not Complete Treatment	21 / 502 (4.2%)	35 / 502 (7.0%)	
10	1	Status at End of 4 Month Continuation Phase	Deaths	5 / 502 (1.0%)	3 / 502 (0.6%)	
11	2	Status Through 24 Month Follow-up	Relapsed	41 / 471 (8.7%)	21 / 458 (4.6%)	0.0622
12	2	Status Through 24 Month Follow-up	Sputum Negative	371 / 471 (78.8%)	368 / 458 (80.3%)	
13	2	Status Through 24 Month Follow-up	Lost to Follow-up	41 / 471 (8.7%)	45 / 458 (9.8%)	
14	2	Status Through 24 Month Follow-up	Deaths	18 / 471 (3.8%)	24 / 458 (5.2%)	

CHECKING IT TWICE

Table and figure programs are independently verified by a different programmer using the same set of specifications and source data. For each table or figure, the production program generates not only an output file, but also a SAS® dataset that stores the analysis results using the standard table macros as shown above. The independent verification program, (not using the standard macros), generates a SAS® dataset that stores analysis results for each table or figure. The verification program will use the logic described in the specifications and produce statistics in a dataset format that mimics the one shown above for the production program.

At the end of each independent verification program, SAS® macros are called to compare the results from production and verification sides and generate a verification report. In the example shown, the compare macro will use SEG, SEGLABL, and LINLABEL as merge keys, and then compare the values of COL1 through COL3 for each matching record.

Here are the verification steps:

1) Comparing the two analysis results datasets generated from production program and verification program for each table or figure
2) Generating verification (discrepancies) report for individual tables or figures
3) Resolving discrepancies between production programs and verification programs
4) Repeating 1) to 3), until all the discrepancies are resolved

seg	match	linlabel	statcol	valid_value	report_value
1	Y	TREATMENT RESPONSE	1	471 / 502 (93.8%)	471 / 502 (93.8%)
1	N	TREATMENT RESPONSE	2	458 / 502 (91.2%)	457 / 502 (91.2%)
1	N	TREATMENT RESPONSE	3	0.2710^	0.2335^
1	Y	NOT CONVERTED	1	5 / 502 (1.0%)	5 / 502 (1.0%)
1	Y	NOT CONVERTED	2	6 / 502 (1.2%)	6 / 502 (1.2%)
1	Y	DID NOT COMPLETE TREATMENT [B]	1	21 / 502 (4.2%)	21 / 502 (4.2%)
1	Y	DID NOT COMPLETE TREATMENT [B]	2	35 / 502 (7.0%)	35 / 502 (7.0%)
1	Y	DEATHS [C]	1	5 / 502 (1.0%)	5 / 502 (1.0%)
1	Y	DEATHS [C]	2	3 / 502 (0.6%)	3 / 502 (0.6%)
2	Y				

This table shows an example of a validation compare where certain results do not match between the production program ('report_value') and the validation program ('valid_value'). When a mismatch occurs within a statistic segment, all of the rows for that segment are shown: this is often useful in diagnosing the source of the mismatch. When all of the rows match (as in segment 2), then the report shows only a summary line with 'MATCH' set to 'Y'. This provides a confirmation that the segment was in fact checked and passed verification.

A SAS® macro performs following activities to generate the single verification report in an Excel file format (see above):

1. Merge production and verification results into one dataset, by segment and line label
2. Compare the two sets of analysis results
3. Flag the discrepancies in the merged data
4. Transpose the results in table columns into rows with a column identifier
5. If a segment of results match between production and verification, a single row will be printed in the verification report

Both production and verification programmers will use the verification report (Figure 5) to identify any causes of the discrepancies and resolve them. The possible causes and solutions are:

1. Specifications are not clearly understood by both programmers (i.e. data selection or sub setting doesn't match)
   → Update the specifications clearly to reduce ambiguity.
2. Values of segment or line label don't match
   → Fix the values of segment or line label.
3. Bugs in either production or verification program or both
   → Debug and update programs, then re-generate outputs.

When both production and verification results match, the report looks something like in Figure 6. There is only one row for each segment that matches, and the only value filled in is the 'MATCH' column, which is set to 'Y'. The automated compare of this table is now considered complete.

Figure 6. Sample Verification Report of Analysis Results Matched between Production and Verification programs

seg	match	linlabel	statcol	valid_value	report_value	comment
1	Y					
2	Y					

AUTOMATED QC REPORT

Once production and verification programs are being created and individual outputs are being generated, another SAS® macro can be used to keep track of the progress of production and verification process at a detailed level or at a summary level. The following tables show examples of the QC status report.

Figure 7. Sample Output of QC Status Report – Detailed Level

Table Number	Title	Program	Production Program Status	Verification Program Status	Validation Status	Visual Review
Figure 1.1	Survival Curves by Disease Phase	f_mort_01	Program does not exist	Program does not exist	Not Available	Completed
Figure 1.1.2	Mean and Standard Deviation of ALT Value Over Time	f_mean_alt	Program exists but inputs do not	Program exists but inputs do not	Not Available	On-going
Table 1.1	Patient Disposition	t_disp_01	Inputs exist but output does not	Inputs exist but output does not	Not Available	Issue(s) found
Table 1.1.2	Patient Demographics	t_demog	Output is older than inputs	Output is older than inputs	Not Current, Matched	Completed
Table 3.1	Overall Summary of Adverse Event	t_ae_01	Output exists and is current	Output exists and is current	Matched	

Here are the steps to generate the detailed level and summary level QC status reports:

- Import the Excel file (=eTOC.xls tab=Reports), where TLF specifications are stored. This is the 'backbone' of the summary, and is used by the summary report to define all of the expected outputs. The count of outputs in this file provides the denominator for the percentages of outputs broken out by status. The TOC file also links report outputs to the names of source files for the purpose of comparing timestamps.
- Read file information from the operating system for all files in the locations defined for analysis datasets, production programs, production results datasets, verification programs, and verification output reports. The timestamps on these files allow the program to determine whether a given output is 'current'. 'Current' means that the output has a later timestamp than any of its immediate inputs (date files and programs). The code that gathers this information is shown below:

```
filename dirbat "dirall.bat";
data _null_ ;
file dirbat notitles ;
length cmd $200 ;
```

```
cmd = 'dir "' || "..\..\ADaM\Derived Data\\*.*" || '" > vsum_adam.lst     '
put cmd ;
cmd = 'dir "' || "..\Programs\\*.*" || '" > vsum_prog.lst     ';
put cmd ;
cmd = 'dir "' || "..\Derived Data\\*.*" || '" > vsum_outputs.lst  ';
put cmd ;
cmd = 'dir "' || "..\Validation\\*.*" || '" > vsum_vprog.lst    ';
put cmd ;
cmd = 'dir "' || "..\Validation Outputs\\*.*" || '" > vsum_vexcel.lst   ';
put cmd ;
run;
x "dirall";
```

- Read in individual verification report spreadsheets for each output, to determine whether the production and verification results matched or not. The program cycles through all of the spreadsheet names found in the verification output directory, opens each file, and checks whether all of the rows in the spreadsheet have the value 'Y' in the 'MATCH' column. If so, the output is considered verified.
- Based on output names from different sources and the creation dates and time stamps, classify outputs according to a) whether all components exist, b) whether they are current, and c) whether they have a validation match.
- Generate detailed report (see Table 6).
- Generate summary (counts and frequency) for each category in 6) (see Figure 8 below).

Figure 8. Sample Output of QC Status Report – Summary Level

When QC is in Progress

	Total (N=156)
Production Program Status	
Program does not exist	20 (12.8%)
Program exists but inputs do not	1 (0.6%)
Inputs exist but output does not	5 (3.2%)
Output is older than inputs	10 (6.4%)
Output exists and is current	129 (82.7%)
Verification Program Status	
Program does not exist	27 (17.3%)
Program exists but inputs do not	1 (0.6%)
Inputs exist but output does not	8 (5.1%)
Not Current, Not Matched	15 (9.6%)
Not Current, Matched	1 (0.6%)
Current, Not Matched	9 (5.9%)
Current, Matched	95 (60.9%)
Visual Review	
Complete	80 (51.3%)
On-going	10 (6.4%)
Issue(s) found	5 (3.2%)

When all the issues are resolved, the summary of QC status report should look something like Figure 9.

Figure 9. Sample Output of QC Status Report – Summary Level

When All QC issues are resolved

	Total (N=156)
Production Program Status	
Program does not exist	0
Program exists but inputs do not	0
Inputs exist but output does not	0
Output is older than inputs	0
Output exists and is current	156 (100%)
Verification Program Status	
Program does not exist	0
Program exists but inputs do not	0
Inputs exist but output does not	0
Not Current, Not Matched	0
Not Current, Matched	0
Current, Not Matched	0
Current, Matched	156 (100%)
Visual Review	
Complete	156 (100%)
On-going	0
Issue(s) found	0

As a side benefit, determining if all programs are 'current' in the production and verification directories makes it possible to dynamically generate a batch file to re-execute all of the programs that actually require a rerun due to a change in the source data or program. The generated batch file for the production directory would look like the one shown below. A generated batch file with no detail rows confirms that all of the outputs are current.

```
REM -------------------------------------------------------------
REM  batch_run_stale.bat
REM  Execute all stale table programs
REM  If this file contains no lines then all outputs are current
REM -------------------------------------------------------------
"C:\Program Files\SAS\SASFoundation\9.2(32-bit)\sas.exe " t_s1t5_hivn.sas -CONFIG
"C:\Program Files\SAS\SASFoundation\9.2(32-bit)\SASV9.CFG"
"C:\Program Files\SAS\SASFoundation\9.2(32-bit)\sas.exe " t_s2t1_hivp.sas -CONFIG
"C:\Program Files\SAS\SASFoundation\9.2(32-bit)\SASV9.CFG"
```

THE FUTURE OF ANALYSIS METADATA

The current standard for DEFINE.XML (version1.0) does not describe analysis results metadata. Nevertheless, this is a topic that has been discussed in CDISC documents and has been proposed as part of the next version of the DEFINE standard. The proposed types of information described in the ADaM version 2.1 document include items such as those shown below:

Description of Analysis Metadata[2]

Metadata Field	*Definition of field*
DISPLAY IDENTIFIER	Unique identifier for the specific analysis display
DISPLAY NAME	Title of display
RESULT IDENTIFIER	Identifies the specific analysis result within a display
PARAM	Analysis parameter
PARAMCD	Analysis parameter code
ANALYSIS VARIABLE	Analysis variable being analyzed
REASON	Rationale for performing this analysis
DATASET	Dataset(s) used in the analysis.
SELECTION CRITERIA	Specific and sufficient selection criteria for analysis subset and / or numerator
DOCUMENTATION	Textual description of the analysis performed
PROGRAMMING STATEMENTS	The analysis syntax used to perform the analysis

For this information to feed into a define.xml that describes analysis results metadata, there will need to be more detailed requirements for the content. In addition, a schema for the define.xml will need to be updated to accommodate analysis results. An updated style sheet will also be needed to provide a display format for this information. These standards and the XML elements needed are not available as of this writing. However, there is a compelling case to be made that analysis results documentation provides important traceability between analysis datasets and the reported results. Current ADaM documents such as the *ADaM Examples in Commonly Used Statistical Analysis Methods, Version 1.0* contain examples of the type of metadata content that the authors consider useful. Although these documents do not prescribe either methods for managing this metadata nor a display format, there is a clear interest in encouraging sponsors to start thinking about how to manage this crucial link in the chain of logic.

Example of Analysis Results Metadata[3]

Metadata Field	Metadata
DISPLAY IDENTIFIER	Table 14-3.01
DISPLAY NAME	Primary Endpoint Analysis: ADAS Cog (11) - Change from Baseline to Week 24 - LOCF
RESULT IDENTIFIER	Pairwise treatment comparisons
PARAM	ADAS-Cog (11) Total Score
PARAMCD	ACTOT11
ANALYSIS VARIABLE	CHG
REASON	Primary efficacy analysis as pre-specified in protocol

[2] See: CDISC Analysis Data Model, Version 2.1 pp. 24 (www.cdisc.org)
[3] See: CDISC Analysis Data Model, Version 2.1 pp. 24 (www.cdisc.org)

DATASET	ADQSADAS
SELECTION CRITERIA	ITTFL='Y' and AVISIT='Week 24' and PARAMCD='ACTOT11'
DOCUMENTATION	Linear model analysis of ADAS-Cog(11) total score change from baseline at Week 24 for pairwise treatment comparisons and adjusted means; missing values imputed using LOCF, Efficacy population. Used randomized treatment as class variable; site group as class variable; and baseline ADAS-Cog score in model.
PROGRAMMING STATEMENTS	PROC GLM; CLASS SITEGR1 TRTP; MODEL CHG = TRTP SITEGR1 BASE; ESTIMATE 'H VS L' TRTP 0 1 -1; ESTIMATE 'H VS P' TRTP -1 1 0; ESTIMATE 'L VS P' TRTP -1 0 1; LSMEANS TRTP / OM STDERR PDIFF CL; RUN;

The inventory approach outlined here can easily provide a basis for analysis results section of define.xml documentation. While the current primary use of the inventory spreadsheet is as automation tool and project management tool, we envision elaborating on this design when a results metadata standard is available. Many elements of the metadata in the list above are already maintained in the inventory spreadsheet: additional fields could be added easily. We would anticipate using the inventory spreadsheet as an additional input to a define.xml generator.

Using the same table inventory spreadsheet to *generate* analyses as well as *document* them is a promising approach. Using a single source for both processes can make it easier to produce and verify reports, and make it easier for the ultimate audience –the statistical reviewer - to understand them.

DISCUSSION

In this paper, we presented techniques to increase programming efficiency of data analysis and control quality of analysis results. Using SAS® macros to compare analysis results from independent programs as well as to track of the production and verification progress can speed up the verification process. In addition to the automation, visual review of the TLFs is an important part of the QC process. Developing a check list for the visual review of outputs can ensure a consistent review and improve the quality of the TLFs.

Acknowledgement

Our special thanks go to Dr. Chad Heilig for his support of this presentation. The US Public Health Service/Tuberculosis Trials Consortium Study 22 information is used in Figures 1, 3, 4 and 5 with permission.

Thank you to Monika Kawohl for comments on the draft version.

References

CDISC Analysis Data Model, Version 2.1 (www.cdisc.org)

ADaM Examples in Commonly Used Statistical Analysis Methods, Version 1.0 (www.cdisc.org)

Contact Information

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

Linda Collins
PharmaStat, LLC
39899 Balentine Drive, Suite 109
Newark, CA 94560
Work Phone: 510 656-2080
lcollins@pharmastat.com

Elizabeth Li
PharmaStat, LLC
39899 Balentine Drive, Suite 109
Newark, CA 94560
Work Phone: 510 656-2080
elizabethli@pharmastat.com

Appendix A. SAS Code

1: Code to generate 'driver' programs from Excel metadata. Input source is the table 'inventory' spreadsheet eTOC.xls. The output is one 'driver' program file per row in the spreadsheet.

```
/*[Import the Excel file (=eTOC.xls tab=Reports), where TLF specifications are stored]*/
PROC IMPORT OUT= WORK.etoc DATAFILE= "..\Metadata\eTOC.xls" DBMS=EXCEL REPLACE;
AEXC;
SHEET="Reports";
GETNAMES=YES;
RUN;


/*[Create the output file name] This section is executed once for each row in the spreadsheet*/
proc printto print = "ts1t5-hivn.sas" new;  BaseProg -Order
run;

options linesize = 200 ;
data _null_ ;
set etoc;
change=put(today(),date9.);
/*[Put a driver program header information]*/
if (baseprog = trim("ts1t5")) and (order = trim("hivn")) and (variant = trim("01")) ;
file print notitles pagesize = 32000 ;
length txt $250 ;
txt ="/** ------------------------------------------------------------------------------**" ;
put @1 txt $200. ;
txt ="** Program:    s1t5-hivn.sas" ;
put @1 txt $200. ;
txt ="** Order:      " || left(trim(order ));
put @1 txt $200. ;
txt ="** Variant:    " || left(trim(variant ));
put @1 txt $200. ;
txt ="** Generated:  " || left(trim(change));
put @1 txt $200. ;
txt =" ";
put @1 txt $200. ;
txt ="** Report No:   " || left(trim(reportno));
put @1 txt $200. ;
if scan(titles,1,'|') ne ' ' then do ;
   txt ="** Title 1:  " || left(trim(scan(titles,1,'|'))) ;
   put @1 txt $200. ;
end ;
if scan(titles,2,'|') ne ' ' then do ;
   txt ="** Title 2:  " || left(trim(scan(titles,2,'|')));
   put @1 txt $200. ;
end ;
  /*[ . . check for additional '|' characters in the titles variable . . .]*/
txt =" ";
put @1 txt $200. ;
```

```
if dataset1 ne ' ' then do ;
 txt ="** Inputfile: " || left(trim(dataset1));
 put @1 txt $200. ;
end ;
```

/*[. . . check for additional dataset2, dataset3,...]*/

```
txt ="** Outputfile: ts1t5-hivn.rtf" ;
put @1 txt $200. ;
txt ="** ------------------------------------------------------------------------------**/" ;
put @1 txt $200. ;
txt =" ";
put @1 txt $200. ;
```

/*[Write a driver program code]*/

```
if dataset1 ne ' ' then do ;
   txt =" ";
   put @1 txt $200. ;
   txt ="data file1 ;" ;
   put @1 txt $200. ;
   txt ="    set " || left(trim(dataset1)) || " ;" ;
   put @1 txt $200. ;
   if select1 ne ' ' then do ;
      if scan(select1,2, '|') ne ' ' then do ;
         txt ="    where (" ;
         put @1 txt $200. ;
         do wrd = 1 to 200 ;
            txt = "        " || scan(select1,wrd, '|') ;
            if txt ne " " then put @1 txt $200. ;
         end ;
         txt = "    ) ;" ;
         put @1 txt $200. ;
      end ;
      else do ;
         txt ="    where (" || left(trim(select1)) || ") ;" ;
         put @1 txt $200. ;
      end ;
   end ;
   if vars1 ne ' ' then do ;
      if length(vars1) < 190 then do ;
      txt ="    keep  " || left(trim(vars1)) || "  ;" ;
      put @1 txt $200. ;
   end ;
   else do x = 1 to length(vars1) ;
      if scan(vars1,x) ne ' ' then do ;
         txt ="    keep  " || left(trim(scan(vars1,x))) || "  ;" ;
         put @1 txt $200. ;
      end ;
   end ;
end ;
txt ="run ; " ;
put @1 txt $200. ;
end ;
```

/*[. . . use similar code for additional datasets and subsets: dataset2, dataset3,...select2, select3, …, vars2, vars3,…]*/

```
txt =" ";
put @1 txt $200. ;
```

```
txt ="%" || left(trim(baseprog)) || " ( " ;
put @1 txt $200. ;
txt ="                order    = " || left(trim(order)) || " , " ;
put @1 txt $200. ;
txt ="                variant  = " || left(trim(variant)) || "   " ;
put @1 txt $200. ;
txt ="               ) ; " ;
put @1 txt $200. ;
txt =" ";
run ;
```