PharmaSUG 2026 - Paper AP-159

Speeding Up Your Validation Process is As Easy As 1, 2 and 3

Alice M Cheng, Johnson & Johnson, Medtech

ABSTRACT

'NOTE: No unequal values were found. All values compared are exactly equal.' Are you excited to see these statements in your SAS® output? After laboring for hours, finally, you got a match! Eureka! You finally did it! But before you pop a champagne and celebrate, please note these statements are necessary but not sufficient to a perfect match! In this paper, the author introduces the use of &SYSINFO, an automatic macro variable generated by PROC COMPARE. This single value held by &SYSINFO enables us to know accurately the result of the comparison. The author will then demonstrate how one can take advantages of its features to speed up the validation of numerous deliverables such as datasets, tables, listings and figures, and how one can use Excel spreadsheet to make the result more readable. Armed with the techniques, speeding up the validation process is really as easy as 1, 2 and 3.

KEYWORDS: PROC COMPARE, Validation, &SYSINFO, Speed Up, Traffic-Lighting, Excel

INTRODUCTION

To ensure the accuracy of a Clinical Study Report (CSR), analysis datasets, tables, listings and figures generated by the production programmer are often validated by a validation programmer. If SAS is the language of choice, the COMPARE procedure offers nice features to facilitate the validation process. The automatic macro variable &SYSINFO generated by the COMPARE procedure enables us to quickly access the result of the validation and avoid the pitfalls of misinterpretation as stated in Horstman and Muller (2013).

- Below are the topics that will be covered in this paper.
- Don't be blindsided by PROC COMPARE
- An Introduction to &SYSINFO
- Interpretation of the value of &SYSINFO
- Generation of a SAS dataset that summarizes the validation result of multiple deliverables, such as all tables in a CSR.
- Color code to alert serious mismatch
- Put validation results in Excel Spreadsheet to make the results more readable.

OVERVIEW

PROC COMPARE is a very powerful procedure in SAS® that allows users to compare two datasets, to compare variables against variables of the same dataset or variables against variables between two datasets. It is relatively simple, yet powerful procedure to use once users have mastered its features. According to SAS® Procedure Guide 9.4, the basic syntax for COMPARE procedure is as follows:

```
PROC COMPARE <option(s)>;
    BY <DESCENDING> variable-1
       <DESCENDING> variable-2 …  <NOTSORTED>;
    ID <DESCENDING> variable-1
       <DESCENDING> variable-2 … <NOTSORTED>;
     VAR variable(s);
     WITH variable(s); run;
```

Program 1. Syntax for PROC COMPARE

PROC COMPARE has numerous options that enable users to list all variables, all observations, control the display of your comparison output, generate output datasets, suppress the writing of observations to

output datasets when all values in the observations are judged to be equal, etc. Usage of ID statement to uniquely identify the observations for comparison is highly recommended. VAR statement enables one to select the variables to be compared. This statement can be useful when we want to concentrate on certain variable(s) in comparison. Since we are mainly interested in comparison of 2 datasets, BY statement and WITH statements are beyond the scope of this paper.
For more information on PROC COMPARE, please refer to *Base SAS® 9.4 Procedures Guide, Seventh Edition, Pages 419-488.*

DON'T GET BLINDSIDED BY PROC COMPARE

The ultimate goal for a validation programmer is to identify and resolve any discrepancies to get a perfect match! So imagine how **excited** a validation programmer would be to see the following statement in the PROC COMPARE output.

```
NOTE: No unequal values were found. All values compared are exactly equal.
```

Eureka! It is a match! Well, these statements are necessary but not sufficient to a perfect match! Have you checked if the same variables are in both datasets? Are all variables of the same type? Do both datasets contain the same observations? The truth is you can have discrepancies in your datasets and still get the forementioned statements in your PROC COMPARE output! As Horstman and Muller (2014) has warned us, 'Don't get Blindsided by PROC COMPARE'.

Consider the following 2 datasets:

CLASS_PROD

SUBJID	SEX	GRADE
101	M	A
102	M	C
103	F	B
104	F	A

CLASS_QC

SUBJID	SEX	SCORE	GRADE
101	1	92	A
102	1	75	C
103	2	88	B
104	2	98	A
105	1	86	B
106	2	89	B

Example 1. CLASS_PROD vs CLASS_QC

Just at first glance, you can see how different these CLASS_PROD and CLASS_QC are!

1. CLASS_PROD has 3 variables and CLASS_QC has 4 variables.
   SCORE exists only in CLASS_QC.
2. CLASS_PROD has 4 observations and CLASS_QC has 6 observations.
3. SEX is a character variable in CLASS_PROD and a numeric variable in CLASS_QC. Let us compare these 2 datasets with PROC COMPARE.

```
proc compare base=CLASS_PROD compare=CLASS_QC LISTVAR LISTOBS;
   id SUBJID;
   title 'CLASS_PROD vs CLASS_QC';
run;
```

Program 2. Use PROC COMPARE to compare CLASS_PROD vs CLASS_QC

(Note that **LISTVAR** option lists all variables found in only one dataset and **LISTOBS** option lists all observations found in only one dataset.)

Below is the result from PROC COMPARE

CLASS_PROD vs CLASS_QC

```
                        The COMPARE Procedure
                Comparison of WORK.CLASS_PROD with WORK.CLASS_QC
                             (Method=EXACT)

                          Data Set Summary                    (1)   (2)

Dataset                    Created          Modified  NVar    NObs

WORK.CLASS_PROD  23MAR26:01:42:37  23MAR26:01:42:37     3       4
WORK.CLASS_QC    23MAR26:01:42:38  23MAR26:01:42:38     4       6


                          Variables Summary

            Number of Variables in Common: 3.
            Number of Variables in WORK.CLASS_QC but not in WORK.CLASS_PROD: 1.   (1)
            Number of Variables with Conflicting Types: 1.
            Number of ID Variables: 1.
```

```
        Listing of Variables in WORK.CLASS_QC but not in WORK.CLASS_PROD

                          Variable  Type  Length

                          SCORE     Num        8   (1)


        Listing of Common Variables with Conflicting Types

             Variable  Dataset          Type  Length

             SEX       WORK.CLASS_PROD  Char       1   (3)
                       WORK.CLASS_QC    Num        8
```

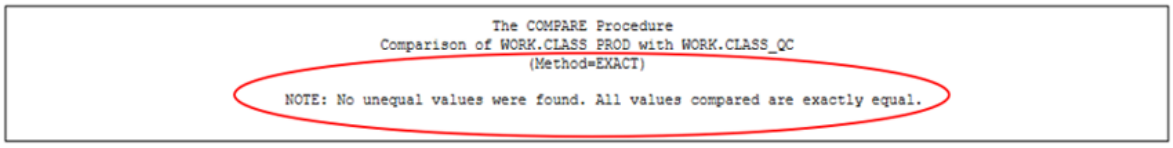
```
                  Comparison Results for Observations

Observation 5 in WORK.CLASS_QC not found in WORK.CLASS_PROD:          (2)
SUBJID=105.

Observation 6 in WORK.CLASS_QC not found in WORK.CLASS_PROD:          (2)
SUBJID=106.
```

```
                          Observation Summary

                    Observation      Base  Compare  ID

                    First Obs           1        1  SUBJID=101
                    Last  Match        4        4  SUBJID=104
                    Last  Obs          .        6  SUBJID=106

Number of Observations in Common: 4.
Number of Observations in WORK.CLASS_QC but not in WORK.CLASS_PROD: 2.
Total Number of Observations Read from WORK.CLASS_PROD: 4.
Total Number of Observations Read from WORK.CLASS_QC: 6.                (2)

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 4.
```

❶ Discrepancies in Number of Variables. Variable SCORE exists only in CLASS_QC.

❷ Discrepancies in Number of Observations. There are 2 more observations in CLASS_QC. And since SUBJID is the ID variable and SUBJID=105 and SUBJID=106 only exist in CLASS_QC, the corresponding observations are not eligible for comparison.

❸ Variable SEX is character in CLASS_PROD, but numeric in CLASS_QC.



Yet, in spite of these discrepancies, we can still see the statements above!

As illustrated by this example, 'NOTE: No unequal values were found. All values compared are exactly equal.' are not reliable statements to identify a perfect match! So how can we identify a perfect match then? &SYSINFO comes to your rescue!

(Note: the author does not think these statements are erroneous, even though they can be misleading! SAS clearly states that 'All values **compared** are exactly equal.' So if there are extra variables, mismatch in types, mismatching in ID variables or extra observations without matching ID variables, SAS cannot do the comparison and hence, not compared. SAS only claimed that the variables SAS could compare were exactly equal; it did not say that the BASE dataset and the COMPARE dataset are perfectly matched!)

WHAT IS &SYSINFO?

&SYSINFO is an automatic SAS macro variable generated by PROC COMPARE to store the return code. The value of the return code provides information about the result of the comparison. Table 1 displays the 16 possible discrepancy conditions. Each condition has a numeric value, displayed in Base 10 and binary 16. format. &SYSINFO is the sum of the values for any of the 16 conditions violated. Please get the return code values immediately after PROC COMPARE, before another procedure or data step. You may want to store the value in another macro variable such as &RC because its value will be reset once another procedurce or data step begins.

```
proc compare base=base data compare=compare data;
run;
%let RC=&SYSINFO;
%put &=RC;
```

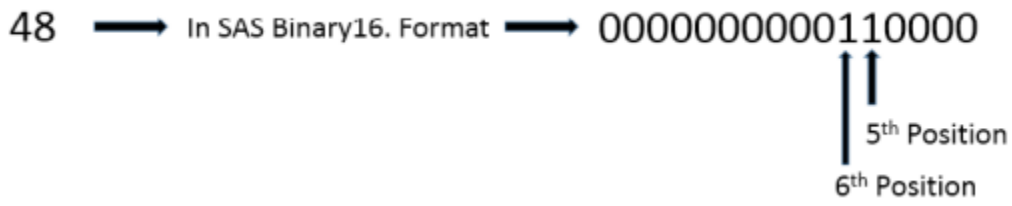
Program 3. PROC COMPARE generates &SYSINFO, whose value is stored in &RC. This value will be displayed in SAS log.

Table 1. The Return Code Values

Bit	Condition	Code	Description	Value Used in binary 16. Format	Value Used in Bit-Testing
1	DSLABEL	$1=2^0$	Data set labels differ.	0000000000000001	'1'b
2	DSTYPE	$2=2^1$	Data set types differ.	0000000000000010	'1.'b
3	INFORMAT	$4=2^2$	Variable has different informat.	0000000000000100	'1..'b
4	FORMAT	$8=2^3$	Variable has different format.	0000000000001000	'1...'b
5	LENGTH	$16=2^4$	Variable has different length.	0000000000010000	'1....'b
6	LABEL	$32=2^5$	Variable has different label.	0000000000100000	'1.....'b
7	BASEOBS	$64=2^6$	Base data set has observation not in comparison.	0000000001000000	'1......'b
8	COMPOBS	$128=2^7$	Comparison data set has observations not in base.	0000000010000000	'1.......'b
9	BASEBY	$256=2^8$	Base dataset has BY group not in comparison.	0000000100000000	'1........'b
10	COMPBY	$512=2^9$	Comparison dataset has BY group not in base.	0000001000000000	'1.........'b
11	BASEVAR	$1024=2^{10}$	Base dataset has variable not in comparison.	0000010000000000	'1..........'b
12	COMPVAR	$2048=2^{11}$	Comparison dataset has variable not in base.	0000100000000000	'1...........'b
13	VALUE	$4096=2^{12}$	A value comparison was unequal.	0001000000000000	'1............'b
14	TYPE	$8192=2^{13}$	Conflicting variables types.	0010000000000000	'1.............'b
15	BYVAR	$16384=2^{14}$	BY variables do not match.	0100000000000000	'1..............'b
16	ERROR	$32768=2^{15}$	Fatal error: comparison not done.	1000000000000000	'1...............'b

HOW TO INTERPRET THE VALUE OF &SYSINFO

&SYSINFO holds a value from 0 to 32788. Each value represents a unique combination of the 16 conditions. Suppose you run the code like those in Program 3 and get the value 48. What does that mean? Which of the 16 conditions are violated? Let us put the number 48 in binary16. format.



The 5th and 6th positions from the right have a value of 1 meaning that the 5th and 6th conditions stated in Table 1 have been violated. In other words, there is at least one variables of different length and at least one variables of different label between the base dataset and compare dataset.

The design of &SYSINFO is truly ingenious! *Putting its value in binary format, one can see that each of the16 conditions is represented by one of the 16 positions! If the condition is true, its value in the corresponding position is a 1; otherwise, a zero. That is why each value of &SYSINFO represents a unique combination of violations! And one single number tells you the result of your comparison!*

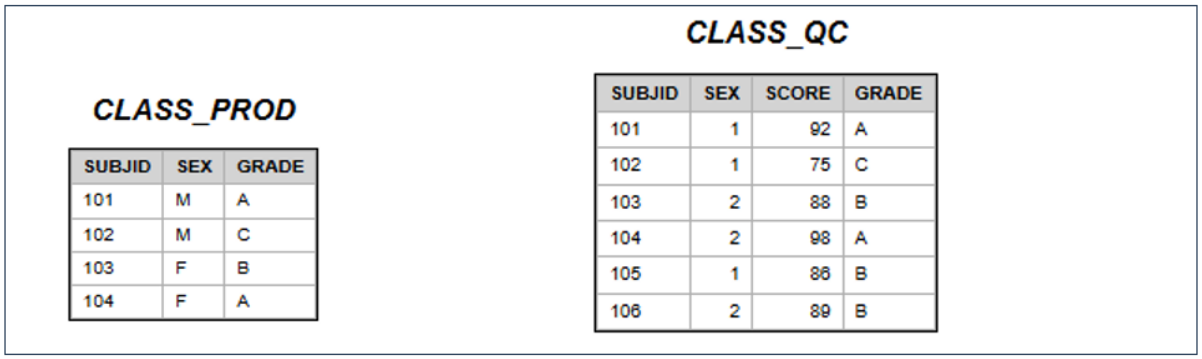
And the ingenuity of &SYSINFO does not stop here! What is the commonality of the first 6 conditions? These conditions are attributes of datasets or variables! Since the maximum value one can get by adding up all the 5 conditions is 63. Any &SYSINFO from 1 to < 64 means different in attributes only. In some cases, difference in attributes may not be that critical! What about &SYSINFO=0? That means none of the 16 conditions have been violated! That is a perfect match! So in a nutshell, the high-level interpretation of &SYSINFO is:

&SYSINFO = 0	means	**Perfect Match!**
&SYSINFO < 64	means	**Difference in Attributes only.**
&SYSINFO >=64	means	**Some Serious Discrepancies!**

Table 2. High Level Interpretation of the value from &SYSINFO.

Is a 'Perfect Match' required? The answer is probably Yes for analysis dataset validation. However, when validating the dataset used for PROC REPORT (One-Proc-Before dataset), difference in attributes may not be that critical. So it all depends what you are validating.

Armed with this knowledge, let us re-consider Example 1.



CLASS_PROD

SUBJID	SEX	GRADE
101	M	A
102	M	C
103	F	B
104	F	A

CLASS_QC

SUBJID	SEX	SCORE	GRADE
101	1	92	A
102	1	75	C
103	2	88	B
104	2	98	A
105	1	86	B
106	2	89	B

Example 1. CLASS_PROD vs CLASS_QC

The following discrepancies have been observed.

1. CLASS_PROD has 3 variables and CLASS_QC has 4 variables.
   SCORE exists only in CLASS_QC.
   (That is Condition 8 in Table 1, having a value of 128.)
2. CLASS_PROD has 4 observations and CLASS_QC has 6 observations.
   (That is Condition 11 in Table 1, having a value of 2048.)
3. SEX is a character variable in CLASS_PROD and a numeric variable in CLASS_QC. Let us compare these 2 datasets with PROC COMPARE.
   (That is Condition 13 in Table 1, having a value of 8192.)

Since &SYSINFO is the sum of all violation condition, the expected value of &SYSINFO will be:

128 + 2048 + 8192 = 10368.

Let double check the result!

```
proc compare base=CLASS_PROD compare=CLASS_QC;
   id SUBJID;
   title 'CLASS_PROD vs CLASS_QC';
run;
%let RC=&SYSINFO;
%put &=RC;
```

Program 4. Get the &SYSINFO value for CLASS_PROD and CLASS_QC comparison.

(Note: The value of &SYSINFO can be directly displayed here without using &RC. However, it may be a good habit to start saving the &SYSINFO value since its value will be reset once another procedure or data step begins.)

CAN THERE BE MORE THAN JUST 16 VIOLATION CONDITIONS

```
71
72          proc compare base=CLASS_PROD compare=CLASS_QC;
73            id SUBJID;
74            title 'CLASS_PROD vs CLASS_QC';
75          run;

NOTE: There were 4 observations read from the data set WORK.CLASS_PROD.
NOTE: There were 6 observations read from the data set WORK.CLASS_QC.
NOTE: PROCEDURE COMPARE used (Total process time):
      real time           0.12 seconds
      cpu time            0.09 seconds


76          %let RC=&sysinfo;
77          %put &=RC;
RC=10368
78
```

&RC has a value of 10368, just as expected.

Program 4 Output: RC=10368 is as expected. It is the total sum of the values for the 3 violation conditions.

One may wonder that given RC=10368, can we easily figure out which conditions have been violated? Although it is not that intuitive, the violation conditions can be easily identified by Bit Map Testing. **Bit Map Testing** checks for the value of 1 in a position, and decides if the corresponding conditions have been violated.

```
%let RC=&SYSINFO;
data _null_;
   /* Test for data set label */
   if &rc = '1'b then
      put '<<<< Data sets have different labels';
   /* Test for data set types */
   If &rc = '1.b' then
      put '<<< Data set types differs';

      ... more similar codes for the other conditions ...
    run;
```

Program 5: Code for Bit Map testing from Base SAS® 9.4 Procedures Guide, Seventh Edition, Page 448.

Based on the code above, the author developed the macro **%INTERPRET** that can easily identify the violation conditions. (Please see %INTERPRET in Appendix 1.) Moreover, result of each violation condition is stored in a macro variable. For instance, if Condition 1 has been violated &RC_1 will be resolved to Y and similarly, for other conditions. So if at the end of Program 4, we add:

```
%INTERPRET (RC=&RC);
```

Program 4 (continue). Invocation of %INTERCEPT

```
177        %interpret(RC=&RC);

<<<< Condition 8: Compare dataset has observation not in base dataset
<<<< Condition 12: Comparison data set has variable not contained in the base data set
<<<< Condition 14: Conflicting variable types between the two data sets being compared
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


RC=10368
RC_1=  RC_2= RC_3= RC_4=
RC_5=  RC_6= RC_7= RC_8=Y
RC_9=  RC_10= RC_11= RC_12=Y
RC_13=  RC_14=Y RC_15= RC_16=
178
```

Program 4 (continue) Output. Invocation of %INTERCEPT result in the SAS Log.

Based on the SAS log above, one can see that for a RC of 10368, Conditions 8, 12 and 14 have been violated. Moreover, the corresponding macro variables for these conditions &RC_8, &RC_12 and &RC_14 all have a value of Y to indicate violation.

Before leaving this section, the author would like to point out that one can also decipher the value of &SYSINFO by means of a function named bAND or bitwise AND. Interested readers should take a look at the paper by Hinson and Coughlin (2012).

CAN THERE BE MORE THAN 16 VIOLATION CONDITIONS

Many years ago, when the author first encountered the &SYSINFO, there were these 16 conditions. Now with Internet, Social Media, AI, SAS Version 9.6m9 and SAS/Viya, we have these 16 conditions! Should more conditions be added?

Scrutinizing over the 16 conditions, the author noted that Condition #15 is 'BY variables do not match'! What about a condition for 'ID variables do not match' ? The world is getting smaller now; sometimes, we get data in different languages. What about another condition on 'Difference in Encoding'? We have encoding displayed in PROC CONTENTS output, why not consider its discrepancy in Encoding in PROC COMPARE? (That said, the author would like to point out that FDA states in "Study Data Technical Conformance Guide, v6.1" Section 3.1.5 that "Use UTF-8 for extending character sets; however, the use of extended mappings is not recommended.")

If these 2 conditions are to be added, should those be Conditions #17 and #18? Note that the current Condition #16 is 'Fatal error: comparison not done'. That can happen if users attempt to compare two entirely different datasets, like you want to compare an Apple to an Orange! I believe this Fatal Error is intentionally placed as the last condition, just like the first 6 conditions are the comparatively mild discrepancies, like difference in data/variable attributes. So can we include these 2 perspective conditions to be Condition #16 and #17 and the current Condition #16 to be Condition #18? What do you think?

The author believes SAS will not add more conditions that contribute to &SYSINFO. And here are the reasons. First, no condition is more fatal than FATAL ERROR. So new conditions cannot be after Condition #16! And additional conditions could not be added before the current Condition #16 because SAS NEEDS TO BE BACKWARD COMPATIBLE! So the author believes these 16 conditions will remain intact! And interested SAS programmers can do their own programming to uncover any additional discrepancies they may be interested in!

EXPAND THE USE OF &SYSINFO

The use of &SYSINFO should not be limited to comparing a pair of datasets. It can be used to validate all analysis datasets or reports in a Clinical Study Report (CSR). In fact, all comparison results can be easily stored in a single dataset so that we can know the matches and mismatches at a glance. Here is how this can be done.

Program 5. Pseudo-code on how result for each program are saved

```
  /*****************************************************************/
  /* Program:    v_t1_demog.sas                                    */
  /* Objective:  To demonstate the layout of each validation       */
  /*             program.                                          */
  /*             Pseudo-Code for Demonstration.                    */
  /*****************************************************************/

  *-----------------------------------------------------------------*;
  *  Initialize work environment.                                   *;
  *  Define libnames.                                               *;
  *-----------------------------------------------------------------*;
  proc datasets lib=work kill nolist nowarn; quit;

  libname sdtm '...';
  libname adam '...';
  libname vout '...';

*-------------------------------------------------------------------*;
*  Step 1                                                           *;
*-------------------------------------------------------------------*;
*  Define macro variables.                                          *;
*-------------------------------------------------------------------
%let pgname=v_t1_pop;
%let tabno=1;
%let title=Basic Demographic Characteristics;
%let pop_text=Full Analysis Set;
%let popfl=FASFL;

%let CUTDATE=2026-02-16;
%let exec_date = %sysfunc(today(), e8601da.);
%let exec_time = %sysfunc(time(), time5.);

*-------------------------------------------------------------------*;
*  Step 2                                                           *;
*-------------------------------------------------------------------*;
*  Create data for comparison.                                      *;
*-------------------------------------------------------------------*;

/*      BODY OF THE PROGRAM HERE.                    */
```

Program 5. Pseudo-code on how result for each program are saved (continue)

```
*------------------------------------------------------------------*;
*  Step 3                                                          *;
*------------------------------------------------------------------*;
*  Perform PROC COMPARE.                                           *;
*------------------------------------------------------------------*;

*--- Perform PROC COMPARE and get the return codes. ---*;
PROC COMPARE BASE=BASE DATASET COMPARE=COMPARE DATASET;
     ID IDVARS;
run;

*--- Remember to get &SYSINFO right after PROC COMPARE or it will be
reset. ---*;
%let RC=&SYSINFO;
%put &=RC;

%INTERPET(RC=&RC);

*------------------------------------------------------------------*;
*  Step 4                                                          *;
*------------------------------------------------------------------*;
*  Store your qc results and related.                              *;
*------------------------------------------------------------------*;

*--- Add result to QC result dataset. ---*;
data qc_&pgname;
   length TABNO $5 TITLE $100 POPULATION $50 PROGRAM $36 STATUS $4
          RC 8. RC_1-RC_16 $3 EXTRACT_DATE $15 QC_EXEC_DATE $15
          QC_EXEC_TIME $15 QC_EXEC_DTM $32 COMMENT $200 COMMENT2
          COMMENT3 $200;

TABNO="&TABNO";
TITLE="&TITLE";
Population="&POP_TEXT";
PROGRAM="&pgname";
RC=&RC;

*--- &RC_1,..., &RC_16 come from invocation of %INTERPRET. ---*;
RC_1=&RC_1; RC_2=&RC_2; RC_3=&RC_3;  … RC_16=&RC_16;

*--- Since these datasets are for the tables, not analysis ---*;
*--- dataset, we can allow some difference in attributes.  ---*;
*--- So &RC < 64 is a PASS here.                           ---*;
*--- If this is for analysis dataset, we may need &RC=0, a ---*;
*--- perfect match in order to pass.                       ---*;

if &RC < 64 then STATUS='PASS';
   else if &RC >= 64 STATUS='FAIL';
```

Program 5. Pseudo-code on how result for each program are saved (continue)

```
  EXTRACT_DATE="&CUTDATE";
  QC_EXEC_DATE="&EXEC_DATE";

  %let len_time=%length(&exec_time);
  %if &len_time le 4 %then %do;
      %let exec_time = 0&exec_time;
  %end;
  QC_EXEC_TIME="&EXEC_TIME";
  QC_EXEC_DTM=catx('T', QC_EXEC_DATE, QC_EXEC_TIME);

 *--- COMMENT, COMMENT2 and COMMENT3 hold values in the title, ---*;
 *--- footnote and column headers.                             ---*;
 *--- These values are not compared in PROC COMPARE. But you   ---*;
 *--- have to make sure that they are correct.                 ---*;
  COMMENT="Drug A: N=&NTRT1; Drug B: N=&NTRT2";
  COMMENT2="Number of Drug A Subjects from Phase 1: &NPH1_TRT1";
  COMMENT3="Number of Drug B Subjects from Phase 1: &NPH1_TRT2 ";

  keep TABNO TITLE POPULATION PROGRAM STATUS RC RC_: QC_EXEC_DTM
       EXTRACT_DATE COMMENT COMMENT2 COMMENT3;
  run;

proc append base=vout.aaa_CSR_results_CSR data=qc_&pgname force;
run;

*--- Keep only the latest result for each program.           ---*;
*--- Each program result is represented by on observation in ---*;
*--- vout.aaa_qc_result_CSR.                                 ---*;
proc sort data=vout.aaa_qc_result_CSR out=vout.aaa_qc_result_CSR;
   by TABNO QC_EXEC_DTM;
run;

data vout.aaa_qc_result_CSR;
  set vout.aaa_qc_result_CSR
  by TABNO QC_EXEC_DTM;
  if last.TABNO;
  run;

*--- End of Program. ---*;
```

Total rows: 1 Total columns: 11 Rows 1-1

	TABNO	TITLE	POPULATION	PROGRAM	RC	STATUS
1	1	Basic Demographic Characteristics	Safety Analysis Set	v_t1_demog.sas	0	PASS

Total rows: 1 Total columns: 11 Rows 1-1

EXTRACT_DATE	QC_EXEC_DTM	COMMENT	COMMENT2
27FEB2026	2026-04-03T19:59	DRUG A: N=246; DRUG B: N=234	Number of Drug A Subjects from Phase 1: 198

Rows 1-1

COMMENT3
Number of Drug B Subjects from Phase 1: 172

Program 5 Output: One observation in vout.aaa_qc_results_csr generated based on the pseudo-code that summarizes the result of the comparison. (RC_1, …, RC_16 are omitted here due to limited space.)

So each program follows the code like the pseudo-code, one will have one dataset summarizing the result of all the reports, like the output below.

Total rows: 6 Total columns: 11 Rows 1-6

	TABNO	TITLE	POPULATION	PROGRAM	RC	STATUS
1	1	Basic Demographic Characteristics	Safety Analysis Set	v_t1_demog.sas	0	PASS
2	2	Primary Efficacy Endpoints	Full Analysis Set	v_t2_prim_eff.sas	16	PASS
3	3	Secondary Efficacy Endpoints	Full Analysis Set	v_t3_sec_eff.sas	16	PASS
4	4	Adverse Events	Safety Analysis Set	v_t4_ae.sas	36	PASS
5	5	Serious Adverse Events	Safety Analysis Set	v_t5_sae.sas	36	PASS
6	6	Lab Results	Safety Analysis Set	v_t6_lab.sas	4096	FAIL

Total rows: 6 Total columns: 11 Rows 1-6

EXTRACT_DATE	QC_EXEC_DTM	COMMENT	COMMENT2	COMMENT3
27FEB2026	2026-04-03T19:59	DRUG A: N=246; DRUG B: N=234	Number of Drug A Subjects from Phase 1: 198	Number of Drug B Subjects from Phase 1: 172
27FEB2026	2026-04-03T20:34	DRUG A: N=246; DRUG B: N=234		
27FEB2026	2026-04-03T20:30	DRUG A: N=222; DRUG B: N=206		
27FEB2026	2026-04-03T20:28	DRUG A: N=246; DRUG B: N=234		
27FEB2026	2026-04-03T20:20	DRUG A: N=222; DRUG B: N=206		
27FEB2026	2026-04-03T20:25	DRUG A: N=246; DRUG B: N=234		

Summary Output 1: Dataset vout.aaa_qc_result_CSR summarizing the result of the 6 reports.

EXPLORE TRAFFIC-LIGHTING

Based on the result in dataset vout.aaa_qc_result_CSR, we can see that there is one report has a RC=0 indicating a Perfect Match, four reports with RC < 64 indicating some discrepancies in attributes and one report with some serious problem because it has RC > 64. Now we can put this dataset vout.aaa_qc_result_CSR in a report and use Traffic-Lighting techniques to highlight the result of the comparison. The following code can achieve this result.

```
ods rtf file="..\validation\output\qc_result.rtf";

title 'Summary of Validation Result';
footnote j=l "Last Modified: %sysfunc(today(), date9.)
%sysfunc(time(), time5.)";
proc report data=vout.aaa_qc_missing center nowd headline headskip
  split='~'  style(report)={outputwidth=100%}
  style(header)={just=ctopmargin=0.02in bottommargin=0.02in};

   column TABNO TITLE POPULATION PROGRAM EXTRACT_DATE QC_EXEC_DTM
         RC STATUS COMMENT COMMENT2 COMMENT3;

   compute RC;
    if RC.SUM=0 then call define (_col_, "style",
                                  "style={background=green}");
      else if RC.SUM < 64 then call define (_col_, "style",
                                   "style={background=orange}");
      else if RC.SUM >= 64 then call define (_col_, "style",
                                    "style={background=red}");
     endcomp;
 run;
ods rtf close;
```

Program 6. Code for Traffic-Lighting to highlight serious discrepancies in a report

Summary of Validation Results

TABNO	TITLE	POPULATION	PROGRAM	EXTRACT_DATE	QC_EXEC_DTM	RC	STATUS	COMMENT	COMMENT2	COMMENT3
1	Basic Demographic Characteristics	Safety Analysis Set	v_t1_demog.sas	27FEB2026	2026-04-03T19:59	0	PASS	DRUG A: N=246; DRUG B: N=234	Number of Drug A Subjects from Phase 1: 198	Number of Drug B Subjects from Phase 1: 172
2	Primary Efficacy Endpoints	Full Analysis Set	v_t2_prim_eff.sas	27FEB2026	2026-04-03T20:20	16	PASS	DRUG A: N=222; DRUG B: N=206		
3	Secondary Efficacy Endpoints	Full Analysis Set	v_t3_sec_eff.sas	27FEB2026	2026-04-03T20:30	16	PASS	DRUG A: N=222; DRUG B: N=206		
4	Adverse Events	Safety Analysis Set	v_t4_ae.sas	27FEB2026	2026-04-03T20:25	36	PASS	DRUG A: N=246; DRUG B: N=234		
5	Serious Adverse Events	Safety Analysis Set	v_t5_sae.sas	27FEB2026	2026-04-03T20:28	36	PASS	DRUG A: N=246; DRUG B: N=234		
6	Lab Results	Safety Analysis Set	v_t6_lab.sas	27FEB2026	2026-04-03T20:34	4096	FAIL	DRUG A: N=246; DRUG B: N=234		

Last Modified: 03APR2026 22:37

Program 6 Output. Summary Report with Traffic-Lighting to highlight serious discrepancies

PUT THE RESULT IN AN EXCEL FILE

With so many variables in the dataset, it can be a bit difficult to read. Putting the QC result dataset in a Excel file may be the solution. With Excel output, readers are not limited to the size of the page and can put in more variableS. Moreover, they can easily sort the output, filter the output, freeze the panes to make it easier for them to explore the data.

There are at least 2 ways one can put the result in an Excel file.

(1). PROC EXPORT
(2). ODS EXCEL

ODS EXCEL will be more ideal since it enables us to add style, formats, graphs or the ability to add filters, frozen headers or name the worksheet PROC EXPORT does not have that ability.

Consider the following code.

```
ods EXCEL file="&study\validation\output\qc_results_csr.xlsx";
ods EXCEL options(sheet_name="QC Result" embedded_titles="yes"
                  embedded_footnotes="yes"
                  frozen_headers="yes" frozen_rowheaders="2" );

PROC REPORT …;
   …;
run;
ods EXCEL close;
```

Program 7. Code to put the dataset in an Excel file with first row and first 2 columns frozen and Embedded titles and footnotes.

It is very similar to the code in Program 6 but by this simple ODS statement, we have converted our output to Excel file output. To explore more on Excel output, interested readers may want to read the papers by Sekar (2022) and Mahajani (2019).

CONCLUSION

&SYSINFO from PROC COMPARE is a powerful tool that summarizes the result of each comparison. Based on this single value, one can tell if it is a perfect match, some mismatches in attributes only or some more serious mismatches, such as mismatches in values, variable counts, variable types and observation counts.

By storing the values of &SYSINFO for each comparison in a resulting dataset, one can tell at a glance how each individual comparison is matching. And hence, &SYSINFO can help us speed up the validation process. Moreover, one can use traffic-lighting technique to highlight the result of the comparison. Moreover, by converting aforementioned resulting dataset into an Excel file, one can take advantage of the Excel features to sort, filter, freeze, etc. These features will be the result more readable. Armed with this knowledge, one can speed up the validation process without compromising its quality! And Speeding Up Your Validation Process is *Truly* as Easy as 1, 2 and 3!

REFERENCES

Cheng, A., Wise, M. and Flavin, J. 2016. “How to Speed Up Your Validation Process Without Really Trying”, SAS Global Forum 2016. Available at https://support.sas.com/resources/papers/proceedings16/10840-2016.pdf

Conover, W. 2013, “An Abbreviated PROC COMPARE with Traffic Lighting:, WUSS 2013. Available at https://www.lexjansen.com/wuss/2013/91_Paper.pdf

Food and Drug Administration (CDER, CBER), 2025, “Study Data Technical Conformance Guide: Guidance for Industry Providing Regulatory Submission in Electric Format-Standardized Study Data” (2025). Available at https://www.fda.gov/media/153632/download

Hadden, L., 2006, “See What Traffic-Lighting Can Do For You!”, SUGI 31. Available at https://support.sas.com/resources/papers/proceedings/proceedings/sugi31/142-31.pdf

Hinson, J. and Coughlin, M., 2012 “Deciphering PROC COMPARE Codes: The Use of the bAND Function”, SAS Global Forum 2012. Available at https://support.sas.com/resources/papers/proceedings12/063-2012.pdf

Horstman, J., Muller, R., 2014. “Don’t Get Blindsided by PROC COMPARE”, SAS Global Forum 2014. Available at https://support.sas.com/resources/papers/proceedings14/1615-2014.pdf

Mahajani, M., 2019. “SAS outputs in Excel workbook using ODS Excel”, Phuse EU 2019. Available at https://www.lexjansen.com/phuse/2019/ct/CT09.pdf

SAS Institute Inc., SUPPORT / SAMPLES & SAS NOTES. Sample 45011: Using return codes from automatic macro variable &SYSINFO with PROC COMPARE. Available at: https://support.sas.com/kb/45/011.html

SAS Institute Inc., Base SAS® 9.4 Procedures Guide, Seventh Edition, Pages 419-488. Cary, North Carolina, SAS Publishing. Available at https://documentation.sas.com/api/collections/pgmsascdc/9.4_3.5/docsets/proc/content/proc.pdf?locale=en#nameddest=procwhatsnew94

Sekar, D. (2022), “Excel with ODS EXCEL Destination and PROC REPORT!”, SESUG 2022. Available at https://www.lexjansen.com/sesug/2022/SESUG2022_Paper_232_Final_PDF.pdf

Williams, C., 2010 “PROC COMPARE – Worth Another Look!”, SAS Global Forum 2010. Available at https://support.sas.com/resources/papers/proceedings10/149-2010.pdf

ACKNOWLEDGMENTS

The author would like to express her sincere gratitude to Chevell Parker, Kathryn McLawhorn and Russ Tyndall of SAS® Technical Support and Jane Eslinger of Eslinger Enterprises for sharing their expertise. Mayank Singh of Johnson & Johnson, Medtech for his encouragement and technical assistance. Quentin McMullen of Siemens Healthineers for encouraging me to write this paper. She would also like to thank Yonghong Gao and Lei Liu of Johnson & Johnson, Medtech for their support of this paper. Last but not the least, she is eternally grateful to her alma mater, Iowa State University of Science and Technology where she first learned SAS and worked as a research assistant at the Statistical Computing section, and to her cat, Socks, for his companionship when she worked on this paper and other work.

CONTACT INFORMATION

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

Alice M Cheng
Johnson & Johnson, Medtech (Abiomed), Danvers, MA
E-mail: Alicecheng.isu@gmail.com, acheng9@its.jnj.com
Web: https://www.linkedin.com/in/alicemcheng

APPENDIX: %INTERPRET

```
%macro interpret(RC=);
/************************************************************************
/* Filename     :  interpret.sas
/* Project      :  Macro Development
/*
/* Objective    :  To create a macro named %INTERPRET that
/*
/*                  (1). Decipher the value stored in &SYSINFO and
/*                       uncover which of the 16 conditions have been
/*                       violated.
/*
/*                  (2). Create macro varaibles RC_1, ..., RC_16.
/*                       These macro variables contain a value of Y if the
/*                       corresponding condition has been violated.
/*                       Otherwise, these macro variable values will be
/*                       blank.
/*
/* Author       :  Alice Cheng
/* Date Created :  2026-03-26
/*
/* Input Data   :  Input &SYSINFO value to the parameter RC of this macro.
/*
/* Output Data  :  Violation condition (if any) will be displayed in the
/*                 SAS log.
/*                 Values for Return code of each of the 16 codes, stored
/*                 in macro variables RC_1, ..., RC_16 will also be
/*                 displayed in the SAS log.
/*
/*
/************************************************************************/

%global RC_1 RC_2 RC_3 RC_4 RC_5 RC_6 RC_7 RC_8
        RC_9 RC_10 RC_11 RC_12 RC_13 RC_14 RC_15 RC_16;
%let RC_1=; %let RC_2=; %let RC_3=; %let RC_4=;
%let RC_5=; %let RC_6=; %let RC_7=; %let RC_8=;
%let RC_9=; %let RC_10=; %let RC_11=; %let RC_12=;
%let RC_13=; %let RC_14=; %let RC_15=; %let RC_16=;

data _null_;
 /* Condition 1: Test for data set label */
 if &rc = '1'b then do;
    put '<<<< Conditon 1: Data sets have different labels';
    call symputx('RC_1', 'Y');
 end;
/* Condition 2: Test for data set types */
 if &rc = '1.'b then do;
    put '<<<< Condition 2: Data set types differ';
    call symputx('RC_2', 'Y');
  end;
```

APPENDIX: %INTERPRET (CONTINUE)

```
/* Condition 3: Test for variable informat */
 if &rc = '1..'b then do;
    put '<<<< Condition 3: Variable has different informat';
    call symputx('RC_3', 'Y');
  end;
/* Condition 4: Test for variable format */
 if &rc = '1...'b then do;
    put '<<<< Condition 4: Variable has different informat';
    call symputx('RC_4', 'Y');
  end;
/* Condition 5: Test for length */
 if &rc = '1....'b then do;
    put '<<<< Condition 5: Variable has different lengths';
    call symputx ('RC_5', 'Y');
 end;
/* Condition 6: Test for label */
 if &rc = '1.....'b then do;
    put '<<<< Condition 6: Variable has different label';
    call symputx ('RC_6', 'Y');
 end;
/* Condition 7: Test if base dataset has observation not in compare dataset
*/
 if &rc = '1......'b then do;
    put '<<<< Condition 7: Base dataset has observation not in compare
dataset';
    call symputx ('RC_7', 'Y');
 end;
/* Condition 8: Test if compare dataset has observation not in base dataset
*/
 if &rc = '1.......'b then do;
    put '<<<< Condition 8: Compare dataset has observation not in base
dataset';
    call symputx ('RC_8', 'Y');
 end;
/* Condition 9: Test if base dataset has BY group not in compare dataset */
 if &rc = '1........'b then do;
    put '<<<< Condition 9: Compare dataset has observation not in base
dataset';
    call symputx ('RC_9', 'Y');
 end;
/* Condition 10: Test if compare dataset has BY group not in base dataset
*/
 if &rc = '1.........'b then do;
    put '<<<< Condition 10: Compare dataset has observation not in base
dataset';
    call symputx ('RC_10', 'Y');
 end;
/* Condition 11: Test if Variable in base data set not in compare data set
*/
 if &rc ='1..........'b then do;
   put '<<<< Condition 11: Variable in base data set not found in
comparison data set';
   call symputx ('RC_11', 'Y');
 end;
```

APPENDIX: %INTERPRET (CONTINUE)

```
/* Condition 12: Test if Comparison data set has variable not in base data
set */
 if &rc = '1...........'b then do;
   put '<<<< Condition 12: Comparison data set has variable not contained
in the base data set';
   call symputx ('RC_12', 'Y');
 end;
/* Condition 13: Test for values */
 if &rc = '1............'b then do;
    put '<<<< Condition 13: A value comparison was unequal';
    call symputx ('RC_13', 'Y');
 end;
/* Condition 14: Test for Conflicting variable types */
 if &rc ='1............'b then do;
    put '<<<< Condition 14: Conflicting variable types between the two data
sets being compared';
    call symputx ('RC_14', 'Y');
 end;
/* Condition 15: Test if BY variable do not match */
 if &rc ='1.............'b then do;
    put '<<<< Condition 15: BY variables do not match';
    call symputx ('RC_15', 'Y');
 end;
/* Condition 16: Test for Fatal Error: comparison not done */
 if &rc ='1.............'b then do;
    put '<<<< Condition 16: Fatal Error: comparison not done';
    call symputx ('RC_16', 'Y');
 end;
 /* When none of the 16 conditions are satisfied, we get a Perfect Match.
*/
 if compress("&RC_1 &RC_2 &RC_3 &RC_4 &RC_5 &RC_6 &RC_7 &RC_8 &RC_9 &RC_10
              &RC_11 &RC_12 &RC_13 &RC_14 &RC_15 &RC_16")=" "  then do;
    put '<<< Congratulations!  Perfect Match!';
 end;
run;
```