SCSUG 2018

Automated Validation of Complex Clinical Trials Made Easy

Richann Watson, DataRich Consulting; Josh Horstman, Nested Loop Consulting

ABSTRACT

Validation of analysis datasets and statistical outputs (tables, listings, and figures) for clinical trials is frequently performed by double programming. Part of the validation process involves comparing the results of the two programming efforts. COMPARE procedure output must be carefully reviewed for various problems, some of which can be fairly subtle. In addition, the program logs must be scanned for various errors, warnings, notes, and other information that might render the results suspect. All of this must be performed repeatedly each time the data is refreshed or a specification is changed. In this paper, we describe a complete, end-to-end, automated approach to the entire process that can improve both efficiency and effectiveness.

INTRODUCTION

In the pharmaceutical industry, the majority of the outputs produced are verified (QC'd) using a double programming process. Double programming requires two programmers to work independently to produce the output based on the provided specifications. Production (PRD) programmer is the individual that is responsible for producing the final output for the deliverable. In the case of table, listing or figure (TLF), the PRD programmer is responsible for not only producing the results but displaying the results in a manner that aligns with the TLF shell mock-ups. Quality Control / Verification (VER) programmer is the individual that is responsible for verifying that the results of the final output are correct. If the final output is a TLF, then the VER programmer also ensures that the output meets cosmetic specifications (i.e., titles, footnotes, indentations, etc. are per the requirements). As an independent programmer, the VER programmer does not consult with the PRD programmer until initial programming is completed from both individuals. Only after initial programming has been completed and a comparison of the PRD and VER outputs has been made should a discussion take place to try and rectify any differences found. One approach to verification of the outputs is for both the PRD and VER programmers to produce permanent data sets of the results so that the VER programmer can run a PROC COMPARE. Although the comparison of two data sets removes a lot of the manual comparison of numbers when QC'ing TLFs, there are still some checks that need to be done by hand. This paper will walk you through the process of producing a permanent production data set that can be used by the VER programmer. In addition, it will talk about some of the pitfalls encountered when doing a manual review of PROC COMPARE; ending with an automated approach to checking the outputs from PROC COMPARE.

PREPARING PRODUCTION OUTPUT

In order to automate the validation process, the PRD programmer needs to produce a permanent data set that can be used by the VER programmer. If the final output is a SAS® data set, then the SAS data set itself is the permanent data set that will be used for verification. However, if the final output is a TLF, then the permanent data set can be produced in one of two ways:

1. Format the data per the TLF specifications and save to a permanent SAS data set. This data set will be used as input into the procedure that will render the final output. There would be no additional formatting or modification to this final data set during the production of the final output. It would be used as is.

2. If using the REPORT procedure, then with the use of the OUT= option the PRD programmer can produce a permanent data set. The OUT = option will produce a data set with a record corresponding to each row of the final output. This includes any blank lines and summary lines. In addition, the OUT = option will produce a variable for every column of the report. When producing these variables SAS will utilize the column name if possible; otherwise it will name the variables based on their positions in the output (e.g., _C1_, _C2_). Thus, it is important to give the variables unique and

meaningful names. Furthermore, the OUT = option will produce an additional variable _BREAK_ that identifies what type of row was generated.

   a.   _BREAK_ equals null indicates a detail row (i.e., a row from the input data set)

   b.   _BREAK_ not equal to null can be based on two different factors.

         i.   If the record is created from a summary, then the value of _BREAK_ is the name of the variable that is used to generate the summary line.

         ii. If the record is created from a COMPUTE BEFORE/AFTER block, then _BREAK_ is the name of the variable used to determine the execution of the compute block. For example, if COMPUTE BEFORE _PAGE_, then _BREAK = '_PAGE_'.

Since the VER programmer will more than likely not use PROC REPORT to produce their output they would either need to manually code these _BREAK_ records into their data set or the PRD programmer can exclude them from the final output by subsetting using a where clause. Refer to SAS Code 1 for an example of PROC REPORT with the OUT = option subsetted based on _BREAK_.

```
libname PRD "directory where SAS data set from PROC REPORT is stored";

proc report data=all split='~' nowindows missing
            out=PRD.RPTDSN (where=(_BREAK_ ne 'ord'));
  column ord population treatment sort status cnt pct;

  define ord        / noprint order order=data;
  define population / order  'Population';
  define treatment  / order  'Treatment';
  define sort       / noprint order order=data;
  define status     / display order=data 'Status~Reason for Exclusion';
  define cnt        / display 'n';
  define pct        / display '%';

  break after ord / page;
run;
```

SAS Code 1. Illustration of PROC REPORT with OUT= option

DOUBLE PROGRAMMING FOR QC

Double programming requires that the VER programmer re-produces the same output that the PRD programmer produced. Regardless whether the final delivery output is a data set or TLF, the production output that will be QC'd is a data set. If the PRD programmer created a permanent data set of the data used to produce the final output, then ideally the VER programmer will create a VER data set with the same variables that was created in production.

After both the PRD and VER programmers have produced their outputs, then the VER programmer would do a PROC COMPARE of the two data sets to see if there were any discrepancies between two. However, once the PROC COMPARE is executed, the output of the comparison needs to be examined.

MANUAL REVIEW OF PROC COMPARE

The most straightforward and common way to assess the outcome of the comparison is a manual review of the listing output generated by the COMPARE procedure. While this is certainly a workable solution, it is tedious, time-consuming, and prone to human error. In large and complex clinical trials involving hundreds of such comparisons, mistakes become inevitable and can jeopardize the integrity of the validation process itself.

When faced with a large number of PROC COMPARE outputs to review, it is tempting for the reviewer to simply look for the following statement at the bottom of the COMPARE output:

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

For the reviewer to definitively conclude from this statement that the data sets are absolutely identical would be a serious error. Horstman and Muller (2014) catalog several potential "blind spots" to this approach:

1. **Missing Variables:** If one data set includes variables not present in the other data set, but the data sets are otherwise identical, PROC COMPARE will still report that "No unequal values were found. All values compared are exactly equal." This statement is misleading but technically true. PROC COMPARE cannot compare variables that don't exist in both data sets, so all the values compared were equal.

2. **Missing Observations:** Similarly, if one data set includes observations not present in the other data set, but the data sets are otherwise identical, PROC COMPARE will once again affirm that "No unequal values were found. All values compared are exactly equal."

3. **Conflicting Types:** If a variable happens to be numeric in one data set and character in the other data set, PROC COMPARE will not be able to compare them. So long as the data set is otherwise identical, PROC COMPARE will make its declaration that "No unequal values were found. All values compared are exactly equal."

4. **Mismatched ID Variables:** If the COMPARE procedure is invoked using the ID statement, then only observations having matching values for the ID variables are compared. Either or both data sets may contain any number of unmatched records that will not be compared. As long as PROC COMPARE finds no discrepancies among the matched records, it will proudly assert that "No unequal values were found. All values compared are exactly equal."

Of course, all these issues are in fact itemized in the full PROC COMPARE output. The problem is that the output is generally so voluminous, especially on a large trial, that it takes a very careful and thorough review to catch every single one. Even worse, this process must be repeated every time the production output is rerun. For these reasons, manual comparison is often not a very practical option.

CREATING A COMPARISON DATA SET

When manual review is not feasible, the next logical step is to automate the review of the PROC COMPARE output. This allows for 100% validation of the *content* of the output. PROC COMPARE has the ability to produce an output data set. Using this data set, the validation of the output can be automated and easily repeated for each potential re-run. For this approach to work the following options need to be utilized:

- **OUT =**: name of the output data set to be created

- **OUTNOEQUAL**: prevents records from being created if the values in the pair of matching observations between BASE and COMP are considered to be equal

- **OUTBASE**: creates a record for each observation in the BASE= data set

- **OUTCOMP**: creates a record for each observation in the COMPARE= data set

- **OUTDIF**: creates a record for each pair of matching observations between BASE and COMP

With the use of these options within PROC COMPARE, a permanent comparison (CMP) data set can be used for checking to see if there were any discrepancies. SAS Code 2 illustrates how PROC COMPARE would be setup to produce the CMP data set.

```
libname PRD "directory where the production data set is stored";
libname VER "directory where the validation data set is stored";
libname CMP "directory where the compare data set is stored";

proc compare base=PRD.RPTDSN (drop=_BREAK_)
             compare=VER.V_RPTDSN listall
             out=CMP.RPTDSN outnoequal outbase outcomp outdif;
  id ord sort;
run;
```

SAS Code 2. Illustration of PROC COMPARE with Various OUT Options

If there are no discrepancies between PRD and VER data sets, then the CMP data set will have zero observations. If there are discrepancies, then the CMP data set will contain records of the following type (i.e., _TYPE_ =)

- **BASE**: either record did not have a matching pair in COMPARE = data set or at least one value between BASE = and COMPARE = was unequal

- **COMPARE**: either record did not have a matching pair in BASE = data set or at least one value between BASE = and COMPARE = was unequal

- **DIF**: shows the difference between the BASE record and the COMPARE record

    o   If the variable is a character variable, the discrepancy will be noted with an 'X'

    o   If the variable is a numeric variable, the discrepancy will be the difference of BASE and COMPARE

Summarizing Results of Comparison Data Set

Once all the VER programmers have completed the programming, the programmer can open the CMP data set to determine if there were any observations and if so then there was a discrepancy between the PRD and VER. However, this can be time consuming if there are numerous data sets to open. An alternative is to run a secondary program that will look at all the CMP data sets to see whether any had more than zero observations. The program would then produce a report that summarizes the results. For more details on this approach, refer to Watson and Johnson (2011).

Limitations with Comparison Data Set

Although it is ideal to want to automate the validation process, the creation of the CMP data set does have its limitations. This approach does require upfront communication between PRD and VER programmers to ensure that the data set structures match (i.e., they need to use the same variable names, lengths and formats). In addition, this process does not produce a record when there are variables in one data set but not the other. Nor does it produce a record if the variable attributes between PRD and VER are different.

PARSING THE PROC COMPARE OUTPUT

To circumvent the limitations of using the OUT options in PROC COMPARE, the .lst files that are typically produced when two data sets are produced can be parsed to look for the various issues that can be encountered in the PROC COMPARE output.

Typical Issues Found in PROC COMPARE

The output from PROC COMPARE is broken down into different sections with each section providing some key information. Some sections only give you basic information about what is being compared and are produced for each PROC COMPARE execution. Other sections are only produced when there is a discrepancy.

In addition, to the types of issues described in Manual Review of PROC COMPARE, SAS Output 1 thru SAS Output 4 illustrate the portions of the COMPARE output that are only provided when there is a discrepancy.

```
            Listing of Common Variables with Differing Attributes

      Variable  Dataset            Type  Length  Label

      AVISIT    WORK.S_ADHD        Char     20
                WORK.V_BIMO_ADHD   Char     16
      ARM       WORK.S_ADHD        Char     32
                WORK.V_BIMO_ADHD   Char     32  Description of Planned Arm
```

SAS Output 1. Listing of Common Variables with Differing Attributes

```
                         Values Comparison Summary

      Number of Variables Compared with All Observations Equal: 54.
      Number of Variables Compared with Some Observations Unequal: 2.
      Number of Variables with Missing Value Differences: 2.
      Total Number of Values which Compare Unequal: 9.
      Maximum Difference: 0.
```

SAS Output 2. Values Comparison Summary

```
                       Variables with Unequal Values

      Variable  Type  Len   Label                    Ndif   MaxDif  MissDif

      TRTDURU   CHAR    3    Treatment Duration Units    8                8
      MISSDOSE  NUM     8    Number of Missed Doses      1       0        1
```

SAS Output 3. Variables with Unequal Values

```
                    Value Comparison Results for Variables


      _____
                     ||  Treatment Duration Units
                     ||  Base Value          Compare Value
               Obs   ||  TRTDURU             TRTDURU
      _____       ||  ___                 ___
                     ||
               27    ||                      DAY
               45    ||                      DAY
               72    ||                      DAY
               75    ||                      DAY
               81    ||                      DAY
               88    ||                      DAY
              108    ||                      DAY
              139    ||                      DAY

      _____
```

SAS Output 4. Value Comparison Results for Variables

Although the other sections are produced for every PROC COMPARE they do have pertinent information that can be used to determine if the data sets match exactly.

5

Automated Solution to Checking Compare Output

As previously noted there are some limitations not only with a manual review of the output but also with the use of the OUT options. A solution to this is the use of the CHECKCMPS macro (

Appendix 1 CHECKCMPS Macro). The macro will parse through each file in the specified directories looking for any issues. The macro can take up to five parameters with only one being required.

- *loc*: Required
  - location(s) where the compare outputs reside
  - if multiple locations are specified, then the locations needed to be separated by the default delimiter (@) or the delimiter that is specified
- *loc2*: Optional
  - location of where the compare summary report will be stored
  - if loc2 is not specified then the location defaults to the first location specified in the loc parameter
- *fnm*: Optional
  - indicates which types of compare files should be parsed
  - if more than one file type then the file types need to be separated with the default delimiter or the delimiter that is specified;
  - if fnm is not specified the all .lst files in the specified location(s) will be parsed
- *delm*: Optional
  - delimiter to be used when specifying multiple locations and/or multiple file types
- *out*: Optional
  - name of the summary compare report

Table 1 illustrates some sample calls of the macro when there is only one directory specified. Table 2 illustrates sample calls when there are multiple directories. Both tables provide an explanation of the expected outcome for each type of call. Note that for macro parameters that are not specified, default values will be used or a default value will be determined as in the case of multiple directories.

Row	Sample Call with One Location Specified	Expected Outcome
1	`%checkcmps(loc=C:\Check Compare\compare output\all,`	- Check **ALL** lst files in the specified directory - Name the summary report "all_checkcmps" - Store summary report in the same directory
2	`%checkcmps(loc=C:\Check Compare\compare output\all,` `        loc2=C:\Check Compare\compare report)`	- Check **ALL** lst files in the specified directory - Name the summary report "all_checkcmps" - Store in location specified by loc2

3	`%checkcmps(loc=C:\Check Compare\compare output\all,`
	`        loc2=C:\Check Compare\compare report,`
	`        fnm=v_t_)`
		- Check **only** lst files that contain 'v_t_' in the file name in the specified directory - Name the summary report "all_checkcmps" - Store summary report in the location specified by loc2

Row	Sample Call with One Location Specified	Expected Outcome

4	`%checkcmps(loc=C:\Check Compare\compare output\all,`
	`        loc2=C:\Check Compare\compare report,`
	`        fnm=v_t_@v_g_)`
		• Check **only** lst files that contain 'v_t_'or 'v_g_' in the file name in the specified directory
		• Name the summary report "all_checkcmps"
		• Store summary report in the location specified by loc2

5	`%checkcmps(loc=C:\Check Compare\compare output\all,`
	`        loc2=C:\Check Compare\compare report,`
	`        fnm=v_t_!v_g_,`
	`        delm=!,`
	`        out=single_table_graph_comps)`
		• Check **only** lst files that contain 'v_t_' or 'v_g_' in the file name in the specified directory
		• Use delimiter specified to separate the types of files
		• Store summary report in the location specified by loc2
		• Provide a specific name for the summary report

Table 1. Sample CHECKCMPS Calls and Expected Outcomes for a Single Directory

Row	Sample Call with Multiple Locations Specified	Expected Outcome

1	`%checkcmps(loc=C:\Check Compare\compare output\adam@`
	`        C:\Check Compare\compare output\graphs@`
	`        C:\Check Compare\compare output\sdtm@`
	`        C:\Check Compare\compare output\tables)`
		• Check **ALL** lst files in all the specified directories
		• Name the summary report "all_checkcmps"
		• Store summary report in the first directory listed in loc (i.e., …\adam)

2	`%checkcmps(loc=C:\Check Compare\compare output\sdtm@`
	`        C:\Check Compare\compare output\adam@`
	`        C:\Check Compare\compare output\graphs@`
	`        C:\Check Compare\compare output\tables)`
		• Same as row 1 but store the report will be stored in …\sdtm since that is the first directory listed

3	`%checkcmps(loc=C:\Check Compare\compare output\adam@`
	`        C:\Check Compare\compare output\graphs@`
	`        C:\Check Compare\compare output\sdtm@`
	`        C:\Check Compare\compare output\tables,`
	`        loc2=C:\Check Compare,`
	`        fnm=v_t_)`
		• Check **only** lst files that contain 'v_t_' in the file name in all the specified directories
		• Name the summary report "all_checkcmps"
		• Store summary report in the first directory specified in loc

4	`%checkcmps(loc=C:\Check Compare\compare output\adam#`
	`        C:\Check Compare\compare output\graphs#`
	`        C:\Check Compare\compare output\sdtm#`
	`        C:\Check Compare\compare output\tables,`
	`        loc2=C:\Check Compare\compare report,`
	`        fnm=v_t_#v_g_,`
	`        delm=#)`
		• Check **only** lst files that contain 'v_t_' or 'v_g_' in the file name in all the specified directories
		• Use delimiter specified to separate the types of files
		• Name the summary report "all_checkcmps"
		• Store summary report in the directory specified in loc2

5	`%checkcmps(loc=C:\Check Compare\compare output\adam#`
	`        C:\Check Compare\compare output\graphs#`
	`        C:\Check Compare\compare output\sdtm#`
	`        C:\Check Compare\compare output\tables,`
	`        loc2=C:\Check Compare\compare report,`
	`        fnm=v_t_#v_g_,`
	`        delm=#,`
	`        out=multiple_table_graph_comps)`
		• Check **only** lst files that contain 'v_t_' or 'v_g_' in the file name in all the specified directories
		• Use delimiter specified to separate the types of files
		• Store summary report in the directory specified in loc2
		• Provide a specific name for the summary report

Table 2. Sample CHECKCMPS Calls and Expected Outcomes for Multiple Directories

Breaking Down the Process

One of the things most people want to know when a new macro/process is introduced is how it works. There are several pieces to the macro which work together to complete the overall process.

Step 1: Determine the Operating Environment

First, the macro needs to know in what operating environment SAS is being run so the appropriate logic can be executed. The macro is able to run in either a Windows or Unix/Linux environment. There are subtle differences between the two environments when executing certain commands in SAS. The macro will handle these differences and adjust accordingly.

During the determination of the environment, two macro variables are created that will be used in the rest of the program:

- ppmcd – represents the pipe command

- slash

If the execution environment is Windows (i.e., &SYSSCP = WIN), the following macro variables are set:

```
%let ppcmd = %str(dir);
%let slash = \;
```

If the execution environment is Unix (i.e., &SYSSCP = LIN X64), the following macro variables are set:

```
%let ppcmd = %str(ls -l);
%let slash = /;
```

Note that if the environment is something other than WIN or LIN X64, then the program will abort and the code will need to be modified to allow for the new environment.

Step 2: Determine What Files to Check

If the macro parameter *fnm* is specified, the macro will determine if there is one type of file to look for or more than one type of file. The macro will extract each file type based on the delimiter that is used. By default the delimiter is '@' and should be used to separate each file type. If a user specifies another delimiter, that delimiter should be used to separate the file types.

Once each file type is extracted from the macro parameter, a where clause is created using the INDEX function for each file type. If the *fnm* parameter is not specified, all lst files in the indicated directories will be checked. Table 3 shows what type of where clause is built based on whether *fnm* is specified.

Row	Sample Call	Where Clause Built
1	`%checkcmps(loc=C:\Check Compare\compare output\all,`	No **where** clause. Macro will check all lst files in the indicated directory

2	`%checkcmps(loc=C:\Check Compare\compare output\sdtm@`
	`         C:\Check Compare\compare output\adam@`
	`         C:\Check Compare\compare output\graphs@`
	`         C:\Check Compare\compare output\tables)`
		No **where** clause. Macro will check all lst files in the indicated directories

3	`%checkcmps(loc=C:\Check Compare\compare output\all,`
	`         loc2=C:\Check Compare\compare report,`
	`         fnm=v_t_)`
		where index(flst, "**v_t_**")

4	`%checkcmps(loc=C:\Check Compare\compare output\adam#`
	`         C:\Check Compare\compare output\graphs#`
	`         C:\Check Compare\compare output\sdtm#`
	`         C:\Check Compare\compare output\tables,`
	`         loc2=C:\Check Compare\compare report,`
	`         fnm=v_t_#v_g_,`
	`         delm=#)`
		where index(flst, "**v_t_**") or index(flst, "**v_g_**")
		
		Note that in this example the delimiter specified '#' so that is used to separate the file types in fnm.

Table 3. Sample CHECKCMPS Calls and Where Clause Built

Step 3: Delete any Leftover Temporary Data sets with Specific Names

Since the process will append data as it loops through each of the directories, it is necessary to delete temporary data sets with specific names that may have been carried over from a previous run.

```
/* need to make sure data sets do not exist before start processing */
proc datasets;
   delete alllsts alluv report report_1 report_2 report_3;
quit;
```

Step 4: Process Each Directory Specified

Now that the environment is determined and the where clause is built to help select the correct files, each directory needs to be processed individually to look for the appropriate files.

A. However, before getting into the main portion of the program and searching the directory for the files, we need to make sure the directory exists. In order to check for the existence of the directory a temporary libname can be created and then the macro variable SYSLIBRC can be used to see if the libname was successfully assigned.

```
/* need to make sure the location exists so create a temp library */
libname templib&g "&lcn";

/* begin looking through each compare file location for specified types */
/* if &SYSLIBRC returns a 0 then path exists                           */
%do %while ("&lcn" ne "" and &syslibrc = 0);
   ...
%end;
```

B. If the path exists, the macro will proceed to the next step which is to read in all the files in the directory currently being processed. This step will utilize the macro variables assigned during the determination of the operating environment to create a pipe command that will read in every file in the directory and store them in a SAS data set.

```
/* need to build pipe directory statement as a macro var  */
/* because the statement requires a series of single and  */
/* double quotes - by building the directory statement    */
/* this allows the user to determine the directory rather */
/* than it being hardcoded into the program               */
/* macro var will be of the form:'dir "directory path" '  */
data _null_;
  libnm = strip("&lcn");
  dirnm = catx(" ", "'", "&ppcmd", quote(libnm), "'");
  call symputx('dirnm', dirnm);
run;

/* read in the contents of the directory containing the lst files */
filename pdir pipe &dirnm lrecl=32727;
data lsts&g (keep = flst fdat ftim filename numtok);
  infile pdir truncover scanover;
  input filename $char1000.;
  ...
run;
```

C. After all the files are read into a SAS data set, the key information (i.e., filename, file date and file time stamps) from the directory line can be extracted. Each portion of the information on the line is considered a token. Table 4 shows how the filename looks when the directory is read in and only the rows with '.lst' in the filename are kept. The illustration is based on the Windows environment. For the Unix environment, filename would look slightly different but the program will adjust accordingly. The numtok variable is used to count the number of tokens in the filename. This helps to fully extract the filename for those situations where the filename may contain a space or other special character that would be seen as a delimiter.

Row	filename
1	06/30/2017 01:09 AM 50,202 no_proc_compare.lst
2	01/05/2017 02:37 AM 5,545 v_adsl.lst
3	01/11/2017 06:42 PM 12,423 v_ad_adeff.lst
4	04/19/2017 10:12 AM 54,740 v_g_bimo_adhd.lst
5	02/21/2017 03:44 PM 11,695 v_g_line_visit_antibody_titer.lst
6	01/05/2017 01:28 PM 4,295 v_sd_ds.lst
7	01/04/2017 08:11 PM 4,525 v_sd_dv.lst
8	02/08/2017 01:56 PM 24,277 v_t_visit_device.lst
9	01/18/2017 11:39 AM 7,120,902 v_t_visit_trta_all03.lst

Row	flst	fdat	ftim	numtok
1	no_proc_compare	30JUN2017	01:09 AM	5
2	v_adsl	05JAN2017	02:37 AM	5
3	v_ad_adeff	11JAN2017	06:42 PM	5
4	v_g_bimo_adhd	19APR2017	10:12 AM	5
5	v_g_line_visit_antibody_titer	21FEB2017	03:44 PM	5
6	v_sd_ds	05JAN2017	01:28 PM	5
7	v_sd_dv	04JAN2017	08:11 PM	5
8	v_t_visit_device	08FEB2017	01:56 PM	5
9	v_t_visit_trta_all03	18JAN2017	11:39 AM	5

Table 4. Files Retrieved and Filename, Date and Time Captured

D. Once all the filenames are extracted, a macro variable is created that will contain all the filenames separated by either the default delimiter or the user-defined delimiter. This macro variable will allow each file to be processed separately looking for PROC COMPARE output and parsing the compare output for key information.

```
/* create a list of lsts, dates, times and store in macro variables */
/* count number of compare files in specified folder retain in macro var */
proc sql noprint;
  select flst,
         fdat,
         ftim,
         count (distinct flst)
         into : currlsts separated by "&delm",
              : currdats separated by " ",
              : currtims separated by "@",
              : cntlsts
  from lsts&g
  %if &fnm ne    %then where &fullwhr; ; /* need to keep extra semicolon */
quit;

/* only loop thru the dir if number of compare file found is > 0 */
%if &cntlsts ne 0 %then %do;  /* begin conditional if &cntlsts ne 0 */

  /* read in each lst file and check various components of PROC COMPARE */
  %let x = 1;
  %let lg = %scan(&currlsts, &x, "&delm");
  %let dt = %scan(&currdats, &x);
  %let tm = %scan(&currtims, &x, '@');
```

```
/* loop thru each compare file in dir and look for undesirable messages */
/* embed &lg in double quotes in case filename has special chars/spaces */
%do %while ("&lg" ne "");
    ...
%end;
```

The macro will create a counter for those instances that a single VER program corresponds to more than one output. The counter will be used to distinguish between the different PROC COMPARE outputs that are encountered in the file. In addition, the macro will parse out the following key information:

- Name of the two data sets being compared
- Data Set Summary
  - Data set label, if applicable
  - Number of observations in each data set
  - Number of variables in each data set
- Variables Summary
  - Number of variables in common between the two data set
  - Number of variables in one data set but not the other
  - Number of variables with conflicting data types
- Common Variables with Differing Attributes
- Observation Summary
  - Number of observations read in each data set
  - Number of observations in one data set but not the other
  - Number of duplicate observations in each data set
  - Number of observations with some compared variables unequal
  - Number of observations with all compared variables equal
  - Indicate if all values are exactly equal or indicate number of values that were not exactly equal
- Values Comparison Summary
  - Number of variables with all observations equal
  - Number of variables with some observations equal
  - Number of values with compare unequal
- Variables with Unequal Values
- Value Comparison Results

Step 4A is repeated for each directory specified and if the directory exists the program will proceed to Steps 4B through 4D.

Step 5: Creating the Report

After all the directories are processed, the information gleaned from parsing the compare output will be checked for discrepancies between the BASE and COMPARE data sets. If a discrepancy is found, a message is produced to indicate there is a difference between the two data sets. If there is no discrepancy, a message will be created that indicates "**BASEDSN** and **COMPDSN** match."

If the *loc2* parameter is specified, then the report will be saved in the location specified; otherwise the report will be saved in the first directory that is specified in *loc* parameter. If the *out* parameter is specified, the report will be saved with the name provided; otherwise the report name will default to 'all_checkcmps'. The report will be displayed by location and will contain the name of the compare output, the date and time the compare output was generated, the PROC COMPARE number (i.e., the counter created in case there is more than one PROC COMPARE within a file), the name of the two data sets being compared, and a description of any discrepancies.

Note that if the VER programmer creates temporary copies of the PRD data sets and uses those in PROC COMPARE, then the temporary data set names will be displayed in the summary report. Therefore, it is important to either use the permanent data sets when doing the comparison or give the temporary data sets meaningful names. Without meaningful names, it is not readily evident what was checked especially if there is more than one compare output in the file. Display 1 provides a sample

report and the portion highlighted in yellow points out the temporary data sets that did not have meaningful names.

Lst Name	Lst Date	Lst Time	PROC COMPARE Number*	BASE Data Set	COMPARE Data Set	PROC COMPARE Message
V_ADSL	05JAN2017	02:37 AM	1	MADAM.ADSL	MADAM.ADSL_ORIG	Number of obs in common (143) between DEV and VER found in Observation Summary Section do not match number of obs in DEV found in Data Set Summary Section.
						In Observation Summary Section, number of obs in common (143) between DEV and VER do not match number of obs read for DEV (152).
						Some values are not equal for variables and observations checked: 9
						Number of Variables Compared with All Observations Equal: 54
						Number of Variables Compared with Some Observations Unequal: 2
						Variable with Unequal Values: TRTDURU
						Variable with Unequal Values: MISSDOSE
V_AD_ADEFF	11JAN2017	06:42 PM	1	WORK.DEV	WORK.QC	Some values are not equal for variables and observations checked: 0
						Number of Variables Compared with All Observations Equal: 59
						Number of Variables Compared with Some Observations Unequal: 0
						Total Number of Values not EXACTLY Equal: 26972
V_G_BIMO_ADHD	19APR2017	10:12 AM	1	WORK.S_ADHD	WORK.V_BIMO_ADHD	WORK.S_ADHD and WORK.V_BIMO_ADHD match.
						There are 9 duplicate observations in WORK.S_ADHD
						There are 9 duplicate observations in WORK.V_BIMO_ADHD
						Common Variable with Different Attributes: AVISIT
						Common Variable with Different Attributes: ARM
V_G_LAB_VISIT	21FEB2017	03:44 PM	1	WORK.DEV11	WORK.QC11	WORK.DEV11 and WORK.QC11 match.
			2	WORK.DEV12	WORK.QC12	WORK.DEV12 and WORK.QC12 match.
			3	WORK.DEV13	WORK.QC13	WORK.DEV13 and WORK.QC13 match.
			4	WORK.DEV14	WORK.QC14	WORK.DEV14 and WORK.QC14 match.
V_SD_DS	05JAN2017	01:28 PM	1	SDAT.DS	SDAT.V_DS	SDAT.DS and SDAT.V_DS match.
			2	SDAT.SUPPDS	SDAT.V_SUPPDS	SDAT.SUPPDS and SDAT.V_SUPPDS match.
V_T_VISIT_TRTA	08FEB2017	01:56 PM	1	WORK.QC	WORK.DEV	Number of vars between DEV and VER for V_T_VISIT_DEVICE do not match. Number of vars for DEV = 13 Number of vars VER = 11
						Number of vars in common (10) between DEV and VER do not match number of vars in DEV.
						Common Variable with Different Attributes: AVISITN
			2	WORK.QC	WORK.DEV	Number of vars between DEV and VER for V_T_VISIT_DEVICE do not match. Number of vars for DEV = 13 Number of vars VER = 11
						Number of vars in common (10) between DEV and VER do not match number of vars in DEV.
						Number of obs in common (0) between DEV and VER found in Observation Summary Section do not match number of obs in DEV found in Data Set Summary Section.
						In Observation Summary Section, number of obs in common (0) between DEV and VER do not match number of obs read for DEV (420).
						Some values are not equal for variables and observations checked: 420
						Number of Variables Compared with All Observations Equal: 9
						Number of Variables Compared with Some Observations Unequal: 1
						Variable with Unequal Values: VARORD
						Common Variable with Different Attributes: AVISITN

Display 1. Sample CHECKCMPS Report

CHECKING THE LOGS

Although all the outputs have passed QC, the job is not necessarily done. Part of the validation process is to ensure that both the PRD and VER programs have executed successfully. One way is to open all the log files and manually scan the logs for various error messages. This can be very time consuming and is prone to human error and can be easily overlooked especially during "crunch" times.

It is not wise to deliver outputs without first looking at the logs. An alternative to opening each log file for both the production side and the verification side is to allow SAS to parse through each log and look for the unwanted log messages and then provide a report of the findings.

The CHECKLOGS macro will execute in either Windows or Unix. Within the macro are standard messages that are searched for in the logs. Below is a list of the standard messages.

- ERROR

- WARNING

- UNINITIALIZED

- NOTE: MERGE

- MORE THAN ONE DATA SET WITH REPEATS OF BY

- VALUES HAVE BEEN CONVERTED

- MISSING VALUES WERE GENERATED AS A RESULT

- INVALID DATA

- INVALID NUMERIC DATA

- AT LEAST ONE W.D FORMAT TOO SMALL

In addition, the user can create a spreadsheet of the list of possible undesirable log messages and these will be searched for in the logs as well. When creating a spreadsheet with the log messages, the name of the file and the name of the column header in the spreadsheet will need to be specified during the call to the macro.

Table 5 describes the 8 macro parameters. Of the 8 parameters only one is required.

Macro Parameter	Required	Description
loc	Yes	Location(s) where the log files are stored. Multiple locations can be specified but they need to be separated by a delimiter.
loc2	No	Location where the report will be stored. If **loc2** is missing, then the report location will default to the first location specified in **loc**.
fnm	No	Indicates which types of files to look at; if more than one type of file is specified, it should be separated by a delimiter. Default is to check all log files in the specified location(s).
delm	No	Delimiter used to separate types of files. Default value is @.
msgf	No	FULL file name (includes location) of spreadsheet where the user specified log messages are stored.
msgs	No	Sheet/tab name in the spreadsheet (**msgf**) that contains the unwanted log messages.
msgv	No	Name of the column in spreadsheet (**msgf**). If there are spaces in the variable name, they need to be converted to underscores ('_') when specifying it as a macro parameter.
out	No	Indicates the name of the output report file that will be produced.

Table 5. Description of CHECKLOGS Macro Parameters

The CHECKLOGS macro is very similar to the CHECKCMPS macro. Steps 1 through 4C of the CHECKCMPS macro are comparable for CHECKLOGS with modifications allowing for the differences in looking for '.log' instead of '.lst'. In addition, the CHECKLOGS macro will allow for user-defined log messages to be stored in a spreadsheet. If a spreadsheet is specified, then the macro will read in the spreadsheet and build additional search criteria that can be used for parsing the logs.

The main difference is at Step 4D where the CHECKCMPS is looking for key information to extract from the files for discerning discrepancies; the CHECKLOGS will look through the logs for the standard unwanted log messages and the user-defined log messages if those are provided.

Like the CHECKCMPS macro, the CHECKLOGS will produce a report listing all the instances in which an unwanted message is encountered.

For more details on the process that allows SAS to check the logs, refer to Watson (2017).

CONCLUSION

Output validation is the necessary evil of a clinical programmer's life. It can be difficult or it can be easy. This paper discussed the manual approach to reviewing the PROC COMPARE results along with some of the 'blindspots' associated with the report. In addition, it discussed an automated approach to checking the PROC COMPARE results by saving that information to a SAS data set that could be scanned, but that too had its drawbacks. Using the CHECKCMPS and CHECKLOGS macros, validation can be efficient, effective, and automatic.

REFERENCES

Horstman, Joshua M. and Roger D. Muller. "Don't Get Blindsided by PROC COMPARE." SAS Global Forum 2014, Paper 1615-2014.
http://support.sas.com/resources/papers/proceedings14/1615-2014.pdf

Watson, Richann and Patty Johnson. "Automated or Manual Validation: Which One is for You?" PharmaSUG 2011, Paper AD01.
http://www.lexjansen.com/pharmasug/2011/AD/PharmaSUG-2011-AD01.pdf

Watson, Richann. "Check Please: An Automated Approach to Log Checking" SAS Global Forum 2017, Paper 1173-2017.
http://support.sas.com/resources/papers/proceedings17/1173-2017.pdf

CONTACT INFORMATION

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

Richann Watson
DataRich Consulting
513-843-4081
richann.watson@datarichconsulting.com
http://www.datarichconsulting.com

Joshua M. Horstman
Nested Loop Consulting
317-815-5899
josh@nestedloopconsulting.com

APPENDIX 1 CHECKCMPS MACRO

```
/* retrieve all the compare files in the specified directory */
%macro checkcmps(loc=,  /* location of where the compare files are stored  */
                        /* can add multiple locations but they need to be  */
                        /* separated by the default delimiter '@' or the   */
                        /* user specified delimiter                        */
               loc2=, /* location of where report is stored (optional)     */
               fnm=,  /* which types of files to look at (optional)        */
                        /* e.g., Tables - t_, Figures - f_, Listings - l_  */
                        /* separate types of files by delimiter indicated in*/
                        /* the delm macro parameter (e.g., t_@f_)           */
               delm=@,/* delimiter used to separate types of files (opt'l)*/
               out=   /* compare report name (optional)                    */);

  /* need to determine the environment in which this is executed   */
  /* syntax for some commands vary from environment to environment */
  %if &sysscp = WIN %then %do;
    %let ppcmd = %str(dir);
    %let slash = \;
  %end;
  %else %if &sysscp = LIN X64 %then %do;
    %let ppcmd = %str(ls -l);
    %let slash = /;
  %end;
  /* end macro call if environment not Windows or Linux/Unix */
  %else %do;
    %put ENVIRONMENT NOT SPECIFIED;
      abort abend;
  %end;

  /* if a filename is specified then build the where clause */
  %if &fnm ne   %then %do;  /* begin conditional if "&fnm" ne "" */
    data _null_;
      length fullwhr $2000.;
      retain fullwhr;

      /* read in each compare file and check for undesired messages */
      %let f = 1;
      %let typ = %scan(&fnm, &f, "&delm");

      /* loop through each type of filename to build the where clause */
      /* embed &typ in double quotes in case filename has special characters/spaces */
      %do %while ("&typ" ne "");  /* begin do while ("&typ" ne "") */

        partwhr = catt("index(flst, '", "&typ", "')");
        fullwhr = catx(" or ", fullwhr, partwhr);

        call symputx('fullwhr', fullwhr);

        %let f = %eval(&f + 1);
        %let typ = %scan(&fnm, &f, "&delm");
      %end;  /* end do while ("&typ" ne "") */
    run;
  %end;  /* end conditional if "&fnm" ne "" */

  /* need to make sure data sets do not exist before start processing */
  proc datasets;
    delete alllsts alluv report report_1 report_2 report_3;
  quit;

  /* need to process each location for compare files separately */
  %let g = 1;
```

15

```sas
%let lcn = %scan(&loc, &g, "&delm");

/* create a default location for report if report location not specified */
%let dloc = &lcn;

/* need to make sure the location exists so create a temp library */
libname templib&g "&lcn";

/* begin looking through each compare file location for specified types */
/* if &SYSLIBRC returns a 0 then path exists                           */
%do %while ("&lcn" ne "" and &syslibrc = 0);  /* begin do while
                                          ("&lcn" ne "" and &syslibrc = 0) */

  /* need to build pipe directory statement as a macro var  */
  /* because the statement requires a series of single and  */
  /* double quotes - by building the directory statement    */
  /* this allows the user to determine the directory rather */
  /* than it being hardcoded into the program               */
  /* macro var will be of the form:'dir "directory path" '  */
  data _null_;
    libnm = strip("&lcn");
    dirnm = catx(" ", "'", "&ppcmd", quote(libnm), "'");
    call symputx('dirnm', dirnm);
  run;

  /* read in the contents of the directory containing the lst files */
  filename pdir pipe &dirnm lrecl=32727;

  data lsts&g (keep = flst fdat ftim filename numtok);
    infile pdir truncover scanover;
    input filename $char1000.;

    length flst $50 fdat ftim $10;

    /* keep only the compare files */
    if index(filename, ".lst");

    /* count the number of tokens (i.e., different parts of filename) */
    /* if there are no spaces then there should be 5 tokens           */
    numtok = countw(filename,' ','q');

    /* parse out the string to get the lst keep only filename */
    /* note on scan function a negative # scans from the right*/
    /* and a positive # scans from the left                   */
    /* need to build the flst value based on number of tokens */
    /* if there are spaces in the lst name then need to grab   */
    /* each piece of the lst name                             */
    /* the first token that is retrieved will have '.lst' and */
    /* it needs to be removed by substituting a blank          */

    /* entire section below allows for either Windows or Unix */
    /*********** WINDOWS ENVIRONMENT ************/
    /* the pipe will read in the information in */
    /* the format of: date time am/pm size file */
    /* e.g. 08/24/2015 09:08 PM 18,498 ae.lst    */
    /*    '08/24/2015' is first token from left */
    /*    'ae.lst' is first token from right    */
    %if &sysscp = WIN %then %do;  /* begin conditional if &sysscp = WIN */
      do j = 5 to numtok;
        tlst = tranwrd(scan(filename, 4 - j, " "),  ".lst", "");
        flst = catx(" ", tlst, flst);
      end;
      ftim = catx(" ", scan(filename, 2, " "), scan(filename, 3, " "));
```

16

```sas
      fdat = put(input(scan(filename, 1, " "), mmddyy10.), date9.);
    %end;  /* end conditional if &sysscp = WIN */

    /***************************** UNIX ENVIRONMENT *****************************/
    /* the pipe will read in the information in the format of: permissions, user,*/
    /* system environment, file size, month, day, year or time, filename         */
    /* e.g. -rw-rw-r-- 1 userid sysenviron 42,341 Oct 22 2015 ad_adaapasi.lst     */
    /*     '-rw-rw-r--' is first token from left                                  */
    /*     'ad_adaapasi.lst' is first token from right                            */
    %else %if &sysscp = LIN X64 %then %do;  /* begin conditional
                                               if &sysscp = LIN X64 */
      do j = 9 to numtok;
        tlst = tranwrd(scan(filename, 8 - j, " "),  ".lst", "");
        flst = catx(" ", tlst, flst);
      end;

      _ftim = scan(filename, 8, " ");

      /* in Unix if year is current year then time stamp is displayed */
      /* otherwise the year last modified is displayed                */
      /* so if no year is provided then default to today's year and if*/
      /* no time is provided indicated 'N/A'                          */
      if anypunct(_ftim) then do;
        ftim = put(input(_ftim, time5.), timeampm8.);
        yr = put(year(today()), Z4.);
      end;
      else do;
        ftim = 'N/A';
        yr = _ftim;
      end;

      fdat = cats(scan(filename, 7, " "), upcase(scan(filename, 6, " ")), yr);
    %end;  /* end conditional if &sysscp = LIN X64 */
run;

/* create a list of lsts, dates, times and store in macro variables */
/* count number of compare files in the specified folder retain in macro var */
proc sql noprint;
  select flst,
         fdat,
         ftim,
         count (distinct flst)
         into : currlsts separated by "&delm",
              : currdats separated by " ",
              : currtims separated by "@",
              : cntlsts
  from lsts&g
  %if &fnm ne   %then where &fullwhr;     ; /* need to keep extra semicolon */
quit;

/* only loop thru the dir if the number of compare file found is greater than 0 */
%if &cntlsts ne 0 %then %do;  /* begin conditional if &cntlsts ne 0 */

  /* read in each lst file and check to see various components of PROC COMPARE */
  %let x = 1;
  %let lg = %scan(&currlsts, &x, "&delm");
  %let dt = %scan(&currdats, &x);
  %let tm = %scan(&currtims, &x, '@');

  /* loop thru each compare file in the dir and look for undesirable messages */
  /* embed &lg in double quotes in case filename has special chars/spaces     */
  %do %while ("&lg" ne "");  /* begin do while ("&lg" ne "") */
    /* read the compare file into a SAS data set to parse the text */
```

```sas
/* need to keep two separate data sets for separate processing */
data lstck&g&x (drop = varvalne)
  uvck&g&x (keep = lst: comp: base: section label varvalne diffattr);
  infile "&lcn.&slash.&lg..lst" truncover pad end=eof;
  input line $char2000.;

  /* need keep a counter of number of comparisons */
  /* in lst file in case there is more than one    */
  if _n_ = 1 then compnum = 0;

  length basedsn compdsn $40 lstlc $200;
  /* add dfsum to see if there are any common vars with different attribs */
  /* add unsum to keep track if there are var with unequal values/results */
  retain basedsn compdsn
         dssum vrsum dfsum obsum vlsum uvsum ursum
         lablpres compnum;

  lstlc = "&lcn";
  label lstlc = 'Compare Location';

  /* need to look for the start of the PROC COMPARE */
  if index(upcase(line), 'THE COMPARE PROCEDURE') then do;
    basedsn = '';
    compdsn = '';
  end;

  /* determine the data sets used for comparison */
  if basedsn = '' and compdsn = '' and
     index(upcase(line), 'COMPARISON OF') and
     index(upcase(line), 'WITH') then do;
    basedsn = scan(line, 3, " ");
    compdsn = scan(line, -1, " ");
  end;

  /* increment number of comparison counter by one every */
  /* time encounter a new 'THE COMPARE PROCEDURE'         */
  if index(upcase(line), 'DATA SET SUMMARY') then do;
    compnum + 1;
  end;

  /* set flags to know which part of PROC COMPARE looking at */
  /* add sct5 to check if there are Variables with Unequal Values */
  /* add sct6 to indicate start of portion where display unequals */
  /* add sct7 to indicate start of differing attributes           */
  /* moved 'SUMMARY' from index statement in macro to part of the */
  /* macro call since not alll sections contain the word 'SUMMARY'*/
  %macro compsect(str =,
                  sct1 =, sct2 =, sct3 =, sct4 =,
                  sct5 =, sct6 =, sct7=);
    if index(upcase(line), "&str") then do;
      &sct1.sum = 1;
      &sct2.sum = .;
      &sct3.sum = .;
      &sct4.sum = .;
      &sct5.sum = .;
      &sct6.sum = .;
      &sct7.sum = .;
    end;
  %mend compsect;

  %compsect(str = DATA SET SUMMARY,
            sct1 = ds, sct2 = vr, sct3 = ob, sct4 = vl,
            sct5 = uv, sct6 = ur, sct7 = df)
```

```
%compsect(str = VARIABLES SUMMARY,
          sct1 = vr, sct2 = ds, sct3 = ob, sct4 = vl,
          sct5 = uv, sct6 = ur, sct7 = df)
%compsect(str = DIFFERING ATTRIBUTES,
          sct1 = df, sct2 = vr, sct3 = ds, sct4 = ob,
          sct5 = vl, sct6 = uv, sct7 = ur)
%compsect(str = OBSERVATION SUMMARY,
          sct1 = ob, sct2 = vr, sct3 = ds, sct4 = vl,
          sct5 = uv, sct6 = ur, sct7 = df)
%compsect(str = VALUES COMPARISON SUMMARY,
          sct1 = vl, sct2 = vr, sct3 = ob, sct4 = ds,
          sct5 = uv, sct6 = ur, sct7 = df)
%compsect(str = VARIABLES WITH UNEQUAL VALUES,
          sct1 = uv, sct2 = ds, sct3 = vr, sct4 = ob,
          sct5 = vl, sct6 = ur, sct7 = df)
%compsect(str = VALUE COMPARISON RESULTS FOR,
          sct1 = ur, sct2 = ds, sct3 = vr, sct4 = ob,
          sct5 = vl, sct6 = uv, sct7 = df)

/* need to determine if there is a label provided on the data sets */
if dssum = 1 and index(upcase(line), 'NVAR') and index(upcase(line), 'NOBS')
   and index(upcase(line), 'LABEL') then lablpres = 'Y';

length section $50 type $10 label $10 value $40 lstnm $25 lstdt lsttm $10;

/* create vars that will contain the compare file that is being scanned */
/* as well as the and date and time that the compare file was created   */
lstnm = upcase("&lg");
lstdt = "&dt";
lsttm = "&tm";

%macro basecomp(type = );
  type = "&type";

  /* determine number of vars and number of observations in each data set */
  if dssum = 1 then do;
    section = 'Data Set Summary';
    /* it is possible for the DEV and QC to have a similar name */
    /* so need to make sure it only matches one so scan for the */
    /* data set name instead of using index                     */
    if strip(scan(line, 1, ' ')) = strip(%upcase(&type.dsn)) then do;
      label = 'NumVar';
      value = scan(line, 4, " ");
      output lstck&g&x;

      label = 'NumObs';
      value = scan(line, 5, " ");
      output lstck&g&x;

      /* if there should be a label then need to build based on tokens */
      if lablpres = 'Y' then do;
        numtok = countw(line, ' ', 'q');
        label = 'DSLbl';
        value = ''; /* need to reset to null so values from
                       previous records are not carried forward */

        do k = 6 to numtok;
          value = catx(" ", value, scan(line, k + 1));
        end; /* end do k = 6 */
        output lstck&g&x;
      end; /* end lablpres = 'Y' */
    end; /* end strip(scan(line, 1, ' '))... */
  end; /* end dssum = 1 */
```

```sas
    /* see how many variabes are in one data set but not other */
    if vrsum = 1 and index(upcase(line), 'NUMBER OF VARIABLES IN') and
       index(upcase(line), 'BUT NOT IN') and
       scan(line, 5, ' ') = strip(%upcase(&type.dsn)) then do;
      section = 'Variables Summary';
      label = 'VarsIn';
      value = scan(line, -1);
      output lstck&g&x;
    end; /* end vrsum = 1 and index(upcase(line), 'NUMBER OF VAR ... */

    /* determine number of observations in common */
    /* it is possible for the DEV and QC to have a similar name */
    /* so need to make sure it only matches one so scan for the */
    /* data set name instead of using index                     */
    if obsum = 1 then do;
      section = 'Observation Summary';
      if index(upcase(line), 'TOTAL NUMBER OF OBSERVATIONS READ') and
         strip(tranwrd(scan(line, 7, ' '), ':', ''))=strip(%upcase(&type.dsn))
         then do;
        label = 'ObsRead';
        value = scan(line, -1);
        output lstck&g&x;
      end; /* end index(upcase(line), 'TOTAL NUMBER ... */
      if index(upcase(line), 'NUMBER OF OBSERVATATIONS IN') and
         index(upcase(line), 'BUT NOT IN') and
         scan(line, 5, ' ') = strip(%upcase(&type.dsn)) then do;
        label = 'ObsIn';
        value = scan(line, -1);
        output lstck&g&x;
      end; /* end index(upcae(line), 'NUMBER OF OBS ... */
      if index(upcase(line), 'NUMBER OF DUPLICATE OBSERVATIONS') and
         strip(tranwrd(scan(line, 7, ' '), ':', ''))=strip(%upcase(&type.dsn))
         then do;
        label = 'DupObs';
        value = scan(line, -1);
        output lstck&g&x;
      end; /* end index(upcase(line), 'NUMBER OF DUP ... */
    end; /* end obsum = 1 */
%mend basecomp;

%basecomp(type = base)
%basecomp(type = comp)

/* these only need to be output once so done outside of macro */
/* see how many variabes are in common and how with have     */
/* different data types (i.e., character versus numeric)      */
if vrsum = 1 then do;
  if index(upcase(line), 'NUMBER OF VARIABLES IN COMMON') then do;
    type = 'Common';
    section = 'Data Set Summary'; /* although this is in Var Summary want to
                                      compare w/values in Data Set Summary */
    label = 'NumVar';
    value = scan(line, -1);
    output lstck&g&x;
  end; /* end index(upcase(line), 'NUMBER OF VAR ... COMMON') */
  if index(upcase(line), 'NUMBER OF VARIABLES WITH CONFLICTING TYPES')
     then do;
    type = 'Conflict';
    section = 'Variables Summary';
    label = 'ConfType';
    value = scan(line, -1);
    output lstck&g&x;
```

```sas
      end; /* end index(upcase(line), 'NUMBER OF VAR ... CONFLICT ... */
   end; /* end vrsum = 1 */

   /* determine number of observations in common */
   if obsum = 1 then do;
      if index(upcase(line), 'COMPARED VARIABLES UNEQUAL') then do;
         type = 'ObsVarNE';
         section = 'Observation Summary';
         label = 'AllEq';
         value = scan(line, -1);
         output lstck&g&x;
      end; /* end index(upcase(line), 'COMPARED VAR ... */
      if index(upcase(line), 'COMPARED VARIABLES EQUAL') then do;
         type = 'Common';
         section = 'Data Set Summary'; /* although this is in Obs Summary want to
                                          compare w/values in Data Set Summary */
         label = 'NumObs';
         value = scan(line, -1);
         output lstck&g&x;

         /* want 3 records for this so it can be compared to */
         /* data set summary as well as the Obs Summary      */
         section = 'Observation Summary';
               label = 'ObsRead';
         output lstck&g&x;

         label = 'AllEq';
         output lstck&g&x;
      end; /* end index(upcase(line), 'COMPARED ... */
      if index(line, 'No unequal values were found.') then do;
         type = 'ObsVarEq';
         section = 'Observation Summary';
         label = 'AllEq';
         value = 'Y';
         output lstck&g&x;
      end; /* end index(line, 'No unequal ... */
   end; /* end obsum = 1 */

   /* determine whether all results are equal or  */
   /* if there are different values for some vars */
   if vlsum = 1 then do;
      if index(upcase(line),
         'NUMBER OF VARIABLES COMPARED WITH ALL OBSERVATIONS EQUAL') then do;
         type = 'AllVarEq';
         section = 'Values Comparison Summary';
         label = 'NumVarVal';
         value = scan(line, -1);
         output lstck&g&x;
      end; /* end index(upcase(line), 'NUMBER OF VAR ... EQUAL') */
      if index(upcase(line),
         'NUMBER OF VARIABLES COMPARED WITH SOME OBSERVATIONS UNEQUAL') then do;
         type = 'SomeVarNE';
         section = 'Values Comparison Summary';
         label = 'NumVarVal';
         value = scan(line, -1);
         output lstck&g&x;
      end; /* end index(upcase(line), 'NUMBER OF VAR ... UNEQUAL') */
      if index(upcase(line),
         'TOTAL NUMBER OF VALUES WITH COMPARE UNEQUAL') then do;
         type = 'ValNE';
         section = 'Values Comparison Summary';
         label = 'NumVarVal';
         value = scan(line, -1);
```

```
        output lstck&g&x;
      end; /* end index(upcase(line), 'TOTAL NUMBER ... */
      if index(upcase(line),
          'TOTAL NUMBER OF VALUES NOT EXACTLY EQUAL') then do;
        type = 'ValNEE';
        section = 'Values Comparison Summary';
        label = 'NumVarVal';
        value = scan(line, -1);
        output lstck&g&x;
      end; /* end index(upcase(line),
            'TOTAL NUMBER OF VALUES NOT EXACTLY EQUAL') */
    end; /* end vlsum = 1 */

    /* need to keep track of all the variables that have unequal values */
    /* this portion will be processed separately so variables will be    */
    /* slightly different than the rest of the variables in this step    */
    if uvsum = 1 then do; /* begin uvsum = 1 */
      if not(index(upcase(line), 'NDIF') or
             index(upcase(line), 'MAXDIF') or
             index(upcase(line), 'VARIABLES WITH UNEQUAL VALUES')) then do;
        section = 'Variables with Unequal Values';
        label = 'VarValNE';
        varvalne = scan(line, 1);
        /* it is possible to have more than one variable with unequal values */
        /* so need to create a counter so that each record is unique         */
        if varvalne ne '' then do;
          uvcnt + 1;
          output uvck&g&x;
        end;
      end;
    end; /*  end uvsum = 1 */

    /* determine if there are variables with different attributes */
    if dfsum = 1 then do; /* begin dfsum = 1 */
      retain colhdr;
      if index(upcase(line), 'VARIABLE') then
        colhdr = index(upcase(line), 'VARIABLE');
      if not(index(upcase(line), 'VARIABLE')) then varloc = anyalpha(line);
      if colhdr = varloc then do;
        section = 'Listing of Common Variables with Differing Attributes';
        label = 'DiffAttr';
        diffattr = scan(line, 1, ' ');
        dfcnt + 1;
        output uvck&g&x;
      end;

    end; /* end dfsum = 1 */
  run;

/* append all the results into one data set */
/* need to check to see if data set exists-during first iteration it should */
/* not exist because it was deleted at top of program                       */
%let exist = %sysfunc(exist(alllsts));
%if &exist = 0 %then %do;
  data alllsts;
    set lstck&g&x;
  run;
%end;
%else %do;
  proc append base=alllsts
              new=lstck&g&x;
  run;
%end;
```

```sas
        /* create a data set that contains all vars with unequal values */
        %let uvexist = %sysfunc(exist(alluv));
        %if &uvexist = 0 %then %do;
          data alluv;
            set uvck&g&x;
          run;
        %end;
        %else %do;
          proc append base=alluv
                      new=uvck&g&x;
          run;
        %end;
        %let x = %eval(&x + 1);
        %let lg = %scan(&currlsts, &x, "&delm");
        %let dt = %scan(&currdats, &x);
        %let tm = %scan(&currtims, &x, '@');
      %end;  /* end do while ("&lg" ne "") */

    %end;  /* end conditional if &cntlsts ne 0 */

    /* end portion to loop through all locations */
    %let g = %eval(&g + 1);
    %let lcn = %scan(&loc, &g, "&delm");

    /* need to make sure the location exists so create a temp library */
    libname templib&g "&lcn";
%end;  /* end do while ("&lcn" ne "" and &syslibrc = 0) */

/* since a list of files can be provided then the files may not be in order */
proc sort data = alllsts presorted;
  by lstlc lstnm lstdt lsttm compnum basedsn compdsn section label type;
run;

/* transpose the data in order to do some checks and create a summary report */
proc transpose data = alllsts
               out = talllsts (drop = _:);
  var value;
  id type;
  by lstlc lstnm lstdt lsttm compnum basedsn compdsn section label;
run;

/* need to determine if certain variables exist */
data _null_;
  dsid = open("talllsts");
  call symputx('confexst', varnum(dsid, 'Conflict'));
  call symputx('vreqexst', varnum(dsid, 'AllVarEq'));
  call symputx('vrneexst', varnum(dsid, 'SomeVarNE'));
  call symputx('vlneexst', varnum(dsid, 'ValNE'));
  call symputx('vneeexst', varnum(dsid, 'ValNEE'));
  rc = close(dsid);

  dsid = open("alluv");
  call symputx('vvneexst', varnum(dsid, 'VarValNE'));
  call symputx('dfatexst', varnum(dsid, 'DiffAttr'));
  rc = close(dsid);
run;

data report_1 (keep = lstlc lstnm lstdt lsttm basedsn compdsn message compnum);
  set talllsts;
  by lstnm lstdt lsttm compnum basedsn compdsn section label;

  length lblmsg varmsg1 varmsg2 varmsg3 varmsg4
```

```
        obsmsg1 obsmsg2 obsmsg3 obsmsg4 obsmsg5 obsmsg6
        aeqmsg dupmsg1 dupmsg2 cnfmsg valmsg1 valmsg2 valmsg3 valmsg4 $200;
retain lblmsg varmsg1 varmsg2 varmsg3 varmsg4
        obsmsg1 obsmsg2 obsmsg3 obsmsg4 obsmsg5 obsmsg6
        aeqmsg dupmsg1 dupmsg2 cnfmsg valmsg1 valmsg2 valmsg3 valmsg4;
if first.compdsn then do;
  lblmsg = '';
  varmsg1 = '';
  varmsg2 = '';
  varmsg3 = '';
  varmsg4 = '';
  obsmsg1 = '';
  obsmsg2 = '';
  obsmsg3 = '';
  obsmsg4 = '';
  obsmsg5 = '';
  obsmsg6 = '';
  aeqmsg = '';
  dupmsg1 = '';
  dupmsg2 = '';
  cnfmsg = '';
  valmsg1 = '';
  valmsg2 = '';
  valmsg3 = '';
  valmsg4 = '';
end;

/* need to check labels and number of vars in compare data sets */
if section = 'Data Set Summary' then do;
  dsn = lstnm;
  if label = 'DSLbl' and base ne '' and comp ne '' and base ne comp then
      lblmsg = catx(" ", "Data set label between DEV and VER for", dsn,
                    "do not match. Label for DEV =", base, "Label for VER =", comp);

  if label = 'NumVar' and base ne '' and comp ne '' and common ne '' then do;
    if base ne comp then
        varmsg1 = catx(" ", "Number of vars between DEV and VER for", dsn,
                       "do not match. Number of vars for DEV =", base,
                       "Number of vars VER =", comp);
    else varmsg1 = '';
    if base ne common then
        varmsg2 = catx(" ", "Number of vars in common (", common,
                       ") between DEV and VER do not match number of vars in DEV.");
    else varmsg2 = '';
  end; /* end label = 'NumVar' and base ne '' ...*/

  /* determine if there are variables in one data set but not the other */
  if label = 'VarsIn' then do;
    if base ne '' then
        varmsg3 = catx(" ", "There are", base, "vars in", basedsn,
                       "that are not in", compdsn);
    else varmsg3 = '';
    if comp ne '' then
        varmsg4 = catx(" ", "There are", comp, "vars in", compdsn,
                       "that are not in", basedsn);
    else varmsg4 = '';
  end; /* end label = 'VarsIn' */
end; /* end section = 'Data Set Summary' */

/* determine if the number of observations between DEV and VER match */
if label = 'NumObs' and base ne '' and comp ne '' and common ne '' then do;
  if base ne comp then obsmsg1 =
    catx(" ","In Data Set Summary Section, number of obs between DEV and VER for",
```

24

```
              dsn, "do not match. Number of obs for DEV =", base,
              "Number of obs VER =", comp);
      else obsmsg1 = '';
      if base ne common then
         obsmsg2 = catx(" ", "Number of obs in common (", common,
                        ") between DEV and VER found in Observation Summary Section do
not match number of obs in DEV found in Data Set Summary Section.");
      else obsmsg2 = '';
   end; /* end label = 'NumObs' ... */

   /* determine if the number of observations read for DEV and VER */
   /* match the number of observations in common                   */
   if label = 'ObsRead' and base ne '' and comp ne '' and common ne '' then do;
      if base ne comp then obsmsg3 = catx(" ",
"In Observation Summary Section, number of observations read for DEV does not match
number of obs read for VER", dsn,
              "Number of obs read for", basedsn, "=", base,
              "Number of obs read for", compdsn, "=", comp);
      else obsmsg3 = '';
      if base ne common then obsmsg4 =
         catx(" ", "In Observation Summary Section, number of obs in common (",
            common,") between DEV and VER do not match number of obs read for DEV (",
            base, ").");
      else obsmsg4 = '';
   end; /* end label = 'ObsRead' ... */

   /* determine if there are observations in one data set but not the other */
   if label = 'ObsIn' then do;
     if base ne '' then obsmsg5 = catx(" ", "There are", base, "obs in", basedsn,
                                       "that are not in", compdsn);
     else obsmsg5 = '';
     if comp ne '' then obsmsg6 = catx(" ", "There are", comp, "obs in", compdsn,
                                       "that are not in", basedsn);
     else obsmsg6 = '';
   end; /* end label = 'ObsIn' */

   if label = 'AllEq' and (obsvareq ne 'Y' or obsvarne ne '0') then aeqmsg =
     catx(" ", "Some values are not equal for variables and observations checked:",
          obsvarne);

   if label = 'DupObs' then do;
     if base ne '' then dupmsg1 = catx(" ", "There are", base,
                                       "duplicate observations in", basedsn);
     else dupmsg1 = '';
     if comp ne '' then dupmsg2 = catx(" ", "There are", comp,
                                       "duplicate observations in", compdsn);
     else dupmsg2 = '';
   end; /* end label = 'DupObs' */

   /* these will only execute if there are issues and */
   /* the PROC COMPARE has unequal values             */
   %if &confexst ne 0 %then %do;
     if label = 'ConfType' then
        cnfmsg = catx(" ", "There are", conflict,
                      "variables with conflicting data types");
   %end;

   %if &vreqexst ne 0 %then %do;
     if label = 'NumVarVal' and allvareq ne . then valmsg1 =
        catx(" ", "Number of Variables Compared with All Observations Equal: ",
             allvareq);
   %end;
```

```sas
    %if &vrneexst ne 0 %then %do;
      if label = 'NumVarVal' and somevarne ne . then valmsg2 =
        catx(" ", "Number of Variables Compared with Some Observations Unequal: ",
             somevarne);
    %end;

    %if &vlneexst ne 0 %then %do;
      if label = 'NumVarVal' and valne ne . then
        valmsg3 = catx(" ", "Total Number of Values with Compare Unequal: ", valne);
    %end;

    %if &vneeexst ne 0 %then %do;
      if label = 'NumVarVal' and valnee ne . then
        valmsg4 = catx(" ", "Total Number of Values not EXACTLY Equal:  ", valnee);
    %end;

    /* want to keep a record for each PROC COMPARE number */
    if last.compnum then do;
      length message $200;
      /* if there are no messages (with exception of duplicates) */
      /* then create a message indicate all matched            */
      /* still need to output duplicate message see array      */
      if cmiss(lblmsg, varmsg1, varmsg2, varmsg3, varmsg4,
               obsmsg1, obsmsg2, obsmsg3, obsmsg4, obsmsg5, obsmsg6,
               aeqmsg, cnfmsg, valmsg1, valmsg2, valmsg3, valmsg4) = /*16*/17 then do;
        message = catx(" ", basedsn, "and", compdsn, "match.");
        output;
      end; /* end cmiss( ... */

      /* create a record for each message so it can be looked into */
      array msg(*) lblmsg varmsg1 varmsg2 varmsg3 varmsg4
                   obsmsg1 obsmsg2 obsmsg3 obsmsg4 obsmsg5 obsmsg6
                   aeqmsg dupmsg1 dupmsg2 cnfmsg valmsg1 valmsg2 valmsg3 valmsg4;
      do i = 1 to dim(msg);
        if msg(i) ne '' then do;
          message = msg(i);
          output;
        end; /* end msg(i) */
      end /* end do i = 1 to dim(msg) */;
    end; /* end last.compdsn */
  run;

/* need to create a comp message of the variables with unequal values */
%if &vvneexst ne 0 %then %do;
  data report_2;
    set alluv;
    where label = 'VarValNE';
    length message $200;
    message = catx(" ", "Variable with Unequal Values: ", varvalne);
  run;
%end;

/* need to create a comp message of the variables with different attributes */
%if &dfatexst ne 0 %then %do;
  data report_3;
    set alluv;
    where label = 'DiffAttr';
    length message $200;
    message = catx(" ", "Common Variable with Different Attributes: ", diffattr);
  run;
%end;

/* combine the all reports into one final report */
```

```
  data report;
    set report_:;
  run;

  /* if the name of the output file is not specified then default to the name */
  %if &out =  %then %do;
    %let out=all_checkcmps;
  %end;

  /* if location of output file is not specified then default to search location */
  %if "&loc2" = "" %then %do;
    data _null_;
      call symputx("loc2", "&dloc");
    run;
  %end;

  /* 20170629 - richann need to sort the data set */
  proc sort data = report presorted;
    by lstlc lstnm;
  run;

  /* create the report */
  ods listing close;
  options orientation=landscape missing = '';

  ods rtf file="&loc2.&slash.&out..rtf";

  title "Summary of PROC COMPARE Results";
  footnote "* If there are multiple PROC COMPARE outputs, the number distinguishes
between the different PROC COMPARE results in the output.";

  /* print by compare file location */
  proc report data=report ls=140 ps=43 spacing=1 missing nowindows headline;
    by lstlc;
    column lstnm lstdt lsttm compnum basedsn compdsn message;
    define lstnm   / order    style(column)=[width=12%]      "Lst Name";
    define lstdt   / order    style(column)=[width=10%]      "Lst Date";
    define lsttm   / order    style(column)=[width=10%]      "Lst Time";
    define compnum / order    style(column)=[width=12%]      "PROC COMPARE Number*";
    define basedsn / order    style(column)=[width=12%]      "BASE Data Set";
    define compdsn / order    style(column)=[width=12%]      "COMPARE Data Set";
    define message / display style(column)=[width=30%] flow "PROC COMPARE Message";

    /* force a blank line after each file */
    compute after lstnm;
      line " ";
    endcomp;
  run;

  ods rtf close;
  ods listing;
%mend checkcmps;
```

APPENDIX 2 CHECKLOGS MACRO

```
/* retrieve all the logs in the specified directory */
%macro checklogs(loc=,   /* location of where the log files are stored     */
                         /* can add multiple locations but they need to be  */
                         /* separated by the default delimiter '@' or the   */
                         /* user specified delimiter                        */
                 loc2=, /* location of where report is stored (optional)   */
                 fnm=,  /* which types of files to look at (optional)      */
```

```sas
                            /* e.g., Tables - t_, Figures - f_, Listings - l_    */
                            /* separate types of files by delimiter indicated in*/
                            /* the delm macro parameter (e.g., t_@f_)            */
                  delm=@,/* delimiter used to separate types of files (opt'l)*/
                  msgf=,  /* FULL file name (includes location) of spreadsheet*/
                            /* where the user specified log messages are stored */
                  msgs=,  /* sheet/tab name in spreadsheet that contains the  */
                            /* unwanted log messages default to 'Sheet1' (opt'l)*/
                  msgv=,  /* name of column in spreadsheet (convert spaces to */
                            /* underscores - must be specified if file specified*/
                            /* (conditionally required)                         */
                  out=   /* log report name (optional)                        */);

/* need to determine the environment in which this is executed    */
/* syntax for some commands vary from environment to environment */
%if &sysscp = WIN %then %do;
  %let ppcmd = %str(dir);
  %let slash = \;
%end;
%else %if &sysscp = LIN X64 %then %do;
  %let ppcmd = %str(ls -l);
  %let slash = /;
%end;
/* end macro call if environment not Windows or Linux/Unix */
%else %do;
    %put ENVIRONMENT NOT SPECIFIED;
      abort abend;
%end;

/* if a filename is specified then build the where clause */
%if "&fnm" ne "" %then %do;  /* begin conditional if "&fnm" ne "" */
  data _null_;
    length fullwhr $2000.;
    retain fullwhr;

    /* read in each log file and check for undesired messages */
    %let f = 1;
    %let typ = %scan(&fnm, &f, "&delm");

    /* loop through each type of filename to build the where clause */
    /* embed &typ in double quotes in case filename has special characters/spaces */
    %do %while ("&typ" ne "");  /* begin do while ("&typ" ne "") */

      partwhr = catt("index(flog, '", "&typ", "')");
      fullwhr = catx(" or ", fullwhr, partwhr);

      call symputx('fullwhr', fullwhr);

      %let f = %eval(&f + 1);
      %let typ = %scan(&fnm, &f, "&delm");
    %end;  /* end do while ("&typ" ne "") */

  run;
%end;  /* end conditional if "&fnm" ne "" */

/* if a spreadsheet is provided with unwanted log messages */
/* then need to use that to build search criteria to be    */
/* used later in the program                               */
%let fullmsg = "";
%if "&msgf" ne "" %then %do;  /* begin conditional if "&msgf" ne "" */
  libname logmsg xlsx "&msgf";

  /* need to make sure spreadsheet exists if it is specified */
```

```sas
   %if %sysfunc(fileexist(&msgf)) = 1 %then %do;  /* begin conditional if
                                                     %sysfunc(filexist(&msgf) = 1 */

     data _null_;
       length fullmsg1 fullmsg2 fullmsg3 $2000;
       retain fullmsg1 fullmsg2 fullmsg3;
       set %if "&msgs" ne "" %then logmsg."&msgs"n;
           %else logmsg."Sheet1"n;
           end = eof; /* need this semicolon to end the set statement */

       partmsg = catt('index(upcase(line), "', (upcase(&msgv)), '")');

       if length(fullmsg1) + length(partmsg) <= 2000 then
           fullmsg1 = catx(" or ", fullmsg1, partmsg);
       else if length(fullmsg2) + length(partmsg) <= 2000 then
           fullmsg2 = catx(" or ", fullmsg2, partmsg);
       else fullmsg3 = catx(" or ", fullmsg3, partmsg);

       if eof then do;
               call symputx('fullmsg1', fullmsg1);
                   if fullmsg2 ne '' then call symputx('fullmsg2', fullmsg2);
                   if fullmsg3 ne '' then call symputx('fullmsg3', fullmsg3);
       end;
     run;
   %end;  /* end conditional if %sysfunc(fileexist(&msgf)) = 1 */

   libname logmsg clear;
 %end;  /* end conditional if "&msgf" ne "" */

 /* need to make sure alllogs does not exist before start processing */
 proc datasets;
   delete alllogs;
 quit;

 /* need to process each location for logs separately */
 %let g = 1;
 %let lcn1 = %scan(&loc, &g, "&delm");

 /* attempt to create a libname for each location specified */
 %do %while ("&&lcn&g" ne "");
   %let g = %eval(&g + 1);
   %let lcn&g = %scan(&loc, &g, "&delm");
 %end;

 /* initialize the location that will act as a place holder till the end of the
program */
 %let dloc=;

 /* loop through each directory specified */
 %do k = 1 %to %eval(&g - 1);  /* begin do i = 1 to %eval(&g - 1) */

     /* need to make sure the location exists so create a temp library */
     libname templib&k "&&lcn&k";

     /* begin looking through each log location for specified log types */
     /* if &SYSLIBRC returns a 0 then path exists                       */
     %if &syslibrc = 0 %then %do;  /* begin if &syslibrc = 0 */

         /* create default location for report if report location not specified */
         /* only create default location for first location that exists          */
         %if "&dloc" = "" %then %do;
             %let dloc = &&lcn&k;
             %end;
```

```sas
/* need to build pipe directory statement as a macro var  */
/* because the statement requires a series of single and  */
/* double quotes - by building the directory statement    */
/* this allows the user to determine the directory rather */
/* than it being hardcoded into the program               */
/* macro var will be of the form:'dir "directory path" '  */
data _null_;
  libnm = "&&lcn&k";
  dirnm = catx(" ", "'", "&ppcmd", quote(libnm), "'");
  call symputx('dirnm', dirnm);
run;

/* read in the contents of the directory containing the logs */
filename pdir pipe &dirnm lrecl=32727;

data logs&k (keep = flog fdat ftim filename numtok);
  infile pdir truncover scanover;
  input filename $char1000.;

  length flog $50 fdat ftim $10;

  /* keep only the logs */
  if index(filename, ".log");

  /* count the number of tokens (i.e., different parts of filename) */
  /* if there are no spaces then there should be 5 tokens          */
  numtok = countw(filename,' ','q');

  /* parse out the string to get the log name */
  /* note on the scan function a negative #   */
  /* scans from the right and a positive #    */
  /* scans from the left                      */
  /* for log keep only first part (e.g. 'ae') */
  /* need to build the flog value based on number of tokens */
  /* if there are spaces in the log name then need to grab  */
  /* each piece of the log name                             */
  /* the first token that is retrieved will have '.log' and */
  /* it needs to be removed by substituting a blank         */
  /* need to do within conditional if statements since num  */
  /* of tokens for Windows is different than Unix           */
  *flog = scan(scan(filename, -1, " "), 1);

  /* entire section below allows for either Windows or Unix */
  /*********** WINDOWS ENVIRONMENT ************/
  /* the pipe will read in the information in */
  /* the format of: date time am/pm size file */
  /* e.g. 08/24/2015 09:08 PM 18,498 ae.log   */
  /*    '08/24/2015' is first token from left */
  /*    'ae.log' is first token from right    */
  %if &sysscp = WIN %then %do;  /* begin conditional if &sysscp = WIN */
    do j = 5 to numtok;
      tlog = tranwrd(scan(filename, 4 - j, " "),  ".log", "");
      flog = catx(" ", tlog, flog);
    end;
    ftim = catx(" ", scan(filename, 2, " "), scan(filename, 3, " "));
    fdat = put(input(scan(filename, 1, " "), mmddyy10.), date9.);
  %end;  /* end conditional if &sysscp = WIN */

  /************************* UNIX ENVIRONMENT ************************/
  /* the pipe will read in information in the format of: permissions, user,*/
  /* system environment, file size, month, day, year or time, filenam      */
  /* e.g. -rw-rw-r-- 1 userid sysenviron 42,341 Oct 22 2015 ad_adaapasi.log*/
```

```sas
      /*     '-rw-rw-r--' is first token from left                   */
      /*     'ad_adaapasi.log' is first token from right             */
      %else %if &sysscp = LIN X64 %then %do;  /* begin conditional if
                                            &sysscp = LIN X64 */

      do j = 9 to numtok;
        tlog = tranwrd(scan(filename, 8 - j, " "),  ".log", "");
        flog = catx(" ", tlog, flog);
      end;

      _ftim = scan(filename, 8, " ");

      /* in Unix if year is current year then time stamp is displayed */
      /* otherwise the year last modified is displayed                */
      /* so if no year is provided then default to today's year and if*/
      /* no time is provided indicated 'N/A'                          */
      if anypunct(_ftim) then do;
        ftim = put(input(_ftim, time5.), timeampm8.);
        yr = put(year(today()), Z4.);
      end;
      else do;
        ftim = 'N/A';
        yr = _ftim;
      end;

      fdat = cats(scan(filename, 7, " "), upcase(scan(filename, 6, " ")), yr);
    %end;  /* end conditional if &sysscp = LIN X64 */

  run;

  /* create a list of logs, dates, times and store in macro variables */
  /* count number of logs in the specified folder and retain in macro variable */
  proc sql noprint;
    select flog,
           fdat,
           ftim,
           count (distinct flog)
           into : currlogs separated by "&delm",
                : currdats separated by " ",
                : currtims separated by "@",
                : cntlogs
    from logs&k
    %if "&fnm" ne "" %then where &fullwhr;
      ; /* need to keep extra semicolon */
  quit;

  /* only loop thru the directory if the number of logs found is greater than 0 */
  %if &cntlogs ne 0 %then %do;  /* begin conditional if &cntlogs ne 0 */

    /* read in each log file and check for undesired messages */
    %let x = 1;
    %let lg = %scan(&currlogs, &x, "&delm");
    %let dt = %scan(&currdats, &x);
    %let tm = %scan(&currtims, &x, '@');

    /* loop thru each log in the directory and look for undesirable messages */
    /* embed &lg in double quotes in case filename has special characters/spaces */
    %do %while ("&lg" ne "");  /* begin do while ("&lg" ne "") */
      /* read the log file into a SAS data set to parse the text */
      data logck&k&x;
        infile "&&lcn&k.&slash.&lg..log" truncover pad;

        /* use $char1000 in order to maintain spacing to get correct line number */
```

```sas
      input line $char1000.;

      /* need to retain the line number so that when a message is encountered */
      /* then will know hwere in log to find it                               */
      retain lineno;

      if _n_ = 1 then lineno = .;
      if anydigit(line) = 1 then
         lineno = input(substr(line, 1, notdigit(line)-1),best.);

      /* keep only the records that had an undesirable message */
      if index(upcase(line), "WARNING") or
         index(upcase(line), "ERROR") or
         index(upcase(line), "UNINITIALIZED") or
         index(upcase(line), "NOTE: MERGE") or
         index(upcase(line), "MORE THAN ONE DATA SET WITH REPEATS OF BY") or
         index(upcase(line), "VALUES HAVE BEEN CONVERTED") or
         index(upcase(line), "MISSING VALUES WERE GENERATED AS A RESULT") or
         index(upcase(line), "INVALID DATA") or
         index(upcase(line), "INVALID NUMERIC DATA") or
         index(upcase(line), "AT LEAST ONE W.D FORMAT TOO SMALL")
         /* allow for user specific messages to be stored in a spreadsheet */
         %if %symexist(fullmsg1) %then or &fullmsg1;
         %if %symexist(fullmsg2) %then or &fullmsg2;
         %if %symexist(fullmsg3) %then or &fullmsg3;
         ; /* need extra semicolon to end if statement */

      /* create variables that will contain the log that is being scanned */
      /* as well as the and date and time that the log file was created   */
      length lognm $25. logdt logtm $10. loglc $200.;
      lognm = upcase("&lg");
      logdt = "&dt";
      logtm = "&tm";

      /* create a dummy variable to create a column on report that will allow */
      /* users to enter a reason if the message is allowed                    */
      logrs = ' ';

      /* need to create a variable that captures the location */
      /* in case there are multiple log locations - need to be*/
      /* print the report by log location                     */
      /* nolog will be used to flag the directories with no   */
      /* logs found                                           */
      loglc = "&&lcn&k";
      label loglc = 'Log Location';
      nolog = .;
   run;

/* because there are sometimes issues with SAS certificate */
/* there will be warnings in the logs that are expected    */
/* these need to be removed                                */
data logck&k&x._2;
   set logck&k&x.;
   if index(upcase(line), 'UNABLE TO COPY SASUSER') or
      index(upcase(line), 'BASE PRODUCT PRODUCT') or
      index(upcase(line), 'EXPIRE WITHIN') or
      (index(upcase(line), 'BASE SAS SOFTWARE') and
       index(upcase(line), 'EXPIRING SOON')) or
      index(upcase(line), 'UPCOMING EXPIRATION') or
      index(upcase(line), 'SCHEDULED TO EXPIRE') or
      index(upcase(line), 'SETINIT TO OBTAIN MORE INFO') then delete;
   run;
```

```sas
      /* determine the number of undesired messages were in the log */
      data _null_;
          if 0 then set logck&k&x._2 nobs=final;
        call symputx('numobs',left(put(final, 8.)));
      run;

      /* if no undesired messages in log create a dummy record for report */
      %if &numobs = 0 %then %do;  /* begin conditional if &numobs = 0 */
        data logck&k&x._2;
          length lognm $25. line $1000. logdt logtm $10. loglc $200.;
          line = "No undesired messages.  Log is clean.";
          lognm = upcase("&lg");
          logdt = "&dt";
          logtm = "&tm";

          /* create a dummy variable to create a column on the report that */
          /* allow users to enter a reason if the message is allowed       */
          logrs = ' ';

          /* adding variables for line number and log location */
          /* nolog will be used to flag the directories with    */
          /* no logs found                                      */
          lineno = .;
          loglc = "&&lcn&k";
          label loglc = 'Log Location';
          nolog = .;
          output;
        run;
      %end;  /* end conditional if &numobs = 0 */

      /* append all the results into one data set */
      /* need to check to see if data set exists-during 1st iteration it should */
      /* not exist because it was deleted at top of program                     */
      %let exist = %sysfunc(exist(alllogs));

      %if /*&x = 1*/ &exist = 0 %then %do;
        data alllogs;
          set logck&k&x._2;
        run;
      %end;
      %else %do;
        proc append base=alllogs
                    new=logck&k&x._2;
        run;
      %end;

      %let x = %eval(&x + 1);
      %let lg = %scan(&currlogs, &x, "&delm");
      %let dt = %scan(&currdats, &x);
      %let tm = %scan(&currtims, &x, '@');
    %end;  /* end do while ("&lg" ne "") */

  %end;  /* end conditional if &cntlogs ne 0 */

  %else %do;
    data nolog&k;
      length lognm $25. line $1000. logdt logtm $10. loglc $200.;
      line = '';
      lognm = '';
      logdt = '';
      logtm = '';
      logrs = '';
      lineno = .;
```

```sas
          loglc = "&&lcn&k";
          label loglc = 'Log Location';
          nolog = 1;
          output;
        run;

        %let exist = %sysfunc(exist(alllogs));

        %if &exist = 0 %then %do;
          data alllogs;
            set nolog&k;
          run;
        %end;
        %else %do;
          proc append base=alllogs
                      new=nolog&k;
          run;
        %end;
      %end;
    %end;   /* end if &syslibrc = 0 */

      %else %do;
        %put %sysfunc(compress(W ARNING:)) "directory &&lcn&k does not exist";
      %end;

%end;  /* begin do i = 1 to %eval(&g - 1) */

/* if the name of the output file is not specified then default to the name */
%if "&out" =  "" %then %do;
  %let out=all_checklogs;
%end;

/* if the name of the output file is not specified then default to the name */
%if "&loc2" = "" %then %do;
  data _null_;
    call symputx("loc2", "&dloc");
  run;
%end;

%let exist = %sysfunc(exist(alllogs));
%if &exist ne 0 %then %do;
      /* since a list of files can be provided then the files may not be in order */
      proc sort data=alllogs presorted;
        by lognm line;
      run;

      /* sort the final report by location */
      proc sort data = alllogs;
        by loglc nolog lognm lineno;
      run;

      /* create the report */
      ods listing close;
      options orientation=landscape missing='';

      ods rtf file="&loc2.&slash.&out..rtf";
      title "Summary of Log Issues";

      /* nolog will determine if message about no */
      /* logs found in directory will be printed  */
      proc report data=alllogs ls=140 ps=43 spacing=1 missing nowindows headline;
        by loglc;
        column nolog lognm logdt logtm lineno line logrs;
```

```sas
      define nolog / analysis sum noprint;
      define lognm / order    style(column)=[width=12%]      "Log Name";
      define logdt / display style(column)=[width=12%]      "Log Date";
      define logtm / display style(column)=[width=12%]      "Log Time";
      define lineno/ display style(column)=[width=12%]      "Line Number";
      define line  / display style(column)=[width=30%] flow "Log Message";
      define logrs / display style(column)=[width=20%] flow "Reason Message is Allowed";

      /* force a blank line after each file */
      compute after lognm;
        line " ";
      endcomp;

      /* if there are no logs in the directory then display message indicating that */
      /* do this to verify that the directory was indeed checked and not overlooked */
      compute after _page_;
        if nolog.sum ^= . then addnote='There are no log files found in the directory';
        line @20 addnote $50.;
      endcomp;
    run;

    ods rtf close;
    ods listing;
  %end;
  %else %do;
      %put %sysfunc(compress(W ARNING:)) "None of the log locations specified exist";
  %end;
%mend checklogs;
```