ABSTRACT

A moderate project in clinical trial programming might have 30-50 ADaM data sets and 200-500 tables, listing, and figures. The convention is 100% independent programming with a comparison of the data sets. When the SAS® System is used, this may be referred to as DP-PC: Double-Programming, Proc COMPARE. Further, a check of the SAS logs for certain words or phrases indicative of unacceptable NOTES, ERROR messages, or WARNING is also appropriate. At times, a lead programmer or biostatistician may want to verify progress or confirm the attestations that Validation is complete yet reviewing so many SAS .lst and .log files manually is cumbersome and a sample may not suffice. The goal of this paper is to introduce five SAS macros and ancillary macros that they require, that submit programs with batch submission, read the contents of a list of directories and optionally provide file metadata, and summarize the results of COMPARE procedures and log checks of all files in directories or single files. Such activities are essential to a readiness audit for delivery or submission and for routine programming.

INTRODUCTION

Validation is an essential component of clinical trial programming. Typically, Validation includes independent (blinded) programming in which specifications and shells are shared, but the code and logs of the Main programmer cannot be viewed by the peer Validation programmer and vice versa. A central task of Validation, besides assuring the accuracy of the Main Programs, includes a verification that the Main programs are of acceptable fidelity as determined by “clean” (SAS® System) logs, that is, the lack of any notes, warnings, or errors that the Standard Operating Procedures (SOPs), Working Instructions (WIs), or the programmers deem unacceptable since these Main programs are part of a submission or delivery and can be subject to audits; they are one of the “faces” of the programmer teams, a point of team pride. While the Main and Validation programmers attest to the successful completion of tasks according to the SOPs/WIs, other audit duties to confirm this must be performed by such people as the lead programmer, the biostatistician, the project managers, or the executives. In fact, some audit tasks, such as confirming that the datetime stamps of the data sets and files are appropriate occurs at the program level at every production execution.

The language can be confusing. A run, submission, or an execution and their respective verbs are synonyms that refer to compiling and executing code. The term submission also refers to body of files composing an application or filing to a regulatory agency (RA, like the FDA) or the act of transferring these files to an RA. A delivery is an instance of a submission of an application or a filing or transferring the package of files (results) outside of the team (group, for instance, transferring Validated and reviewed files to a vice-president or DSMC). This paper refers to program development as the process of drafting or updating code, to QC (Quality Control) as the process of (blinded) peer review and/or programming with Validation programs or code review as the SOPs/WIs require, and to production runs (execution) as the formal execution of code in the appropriate computing environment following the applicable SOPs/WIs. Production runs can be the final step prior to promotion or to produce the final output (see details later in this paragraph), i.e., production runs can occur in the development environment, if one exists. “Formal” or “official” might be better adjectives for this concept, avoiding confusion with the stages below. The Main programmer, sometimes (confusingly) called the Production programmer, has responsibility for creating the output, i.e., the data sets (SDTM or ADaM, for instance) or the PDF and/or RTF files for TFLs (Tables, Figures, and Listings), that will be part of the CSR (Clinical Study Report) and submission. The Validation programmer is the peer programmer who replicates the task typically with 100% independence, but usually does not produce TFL output, and assures that both sets or programs
and output meet the requirements established in the SOPs/WIs. The Main programmer often works in a folder called “PROD” or “SAS”, whereas the Validation programmer often works in a folder called “VAL”. Sometimes, a hierarchy of folders that are nearly replicas of each other distinguish between the stages of development, QC, and final runs and the folders are often named “DEV”, “QC”, and “PROD”, respectively. The Main and Validation programmers both develop in their respective DEV folders, promote acceptable programs to QC for evaluation, and promote programs that pass Validation in QC to PROD for a final run and where other members, like a biostats, may review them. Being in PROD may mean that the product is ready for higher review, but not necessarily delivery; indeed, although individual programs pass Validation, the entire set may not (for instance, a discrepancy in a column total across ten tables may match in Validation, but is not acceptable). Good hygiene might dictate that programmers delete files from lower hierarchical folders upon satisfying requirements for the next level, i.e., a clean run with an acceptable log check and an acceptable match of data sets, which is typically a 100% exact match of values, if not labels, lengths, and attributes. Such a structure and process may not be a requirement, but lack of something similar requires good hygiene and, usually, more details (macro parameter values or post-processing) to use these macros.

At every stage of programming, from development to production runs, the programmers should be checking the datetime stamp of the data sets, files (.sas, .log, .lst) and, if any, raw data files. For instance, the datetime stamp of ADSL should be older than that of any other AdAm data set, but younger than DM (and other required SDTM domains). The datetime stamps of the log and output files should be younger than the (.sas) program that created them. The datetime stamp of the log of the Validation program should be younger than the datetime stamps of the Main log and output. One way to insure this is to have the Validation programmer execute the Main program before executing the Validation program, if the SOPs/WIs allow this and since the Validation programmer should not be opening the Main program (and vice versa), batch submission is a salient and easy solution that also provides an audit trail. A contention might be that ADSL was sent back to development and re-Validated, but ADAE in PROD had not, yet, been re-run and so would any TFL in PROD that used ADAE (and ADSL). Depending on the SOPs/WIs, after completion of programming, i.e., obtaining the attestations of the Main and Validation programmers and passing review by their superiors for each program in the delivery, batch submission of the entire collection of programs, in appropriate hierarchical order, may be required. To “pre-program” a project, then execute every program and deliver within three business days of a database lock is not quite an unusual requirement. After each production execution, every programmer should be checking the log of her or his program prior; no promoted program should generate a log that contains unacceptable phrases that are not documented in the program header (a failure to converge error may be unavoidable, for instance, but the header should alert the programmer and review a priori). If one is a Validator, one should verify that the log of the Main program is acceptable. While this is necessary for each program, the persons responsible for delivery need to confirm that this is true across the entire delivery. While this can be accomplished by spot checking a sample of the programs, programmatically approaching this task is reasonable. Moreover, programmatic approaches may be more accurate and complete, highly efficient, and easily and dependably repeatable.

The goal of this paper is to present SAS System macros 1) to batch submit files using SAS, not the usual .bat approach or another programming languages such as Python, 2) to recursively read the contents of a directory, similar to dir command in DOS or the ls command in Unix/Linux, including, importantly, the reporting of the Last Modified datetime stamps, 3) to summarize a log file from one program with respect to the presence of certain phrases indicating potential or certain unacceptable issues, 4) to summarize log check findings of the logs files in directories, and 5) to summarize the results from COMPARE procedures in single files or every file in a directory. An ancillary macro to delete a list of SAS data sets is included, but like the other macros, can be substituted or modified as the reader sees fit.

TEST CODE AND FILES

Appendix 1 presents a macro for testing and demonstration purposes only. For brevity and clarity, this macro may not follow best practices in a regulated programming environment. For instance, the Validation program needs access to SDTM, but that would be readonly access. The Main programmer should not have to access V_SDTM. In fact, these LIBREFs would not contain the Validation Data Sets (VALDS’s) for TFL programs, but SDTM domains. Lines 11-41 “clean up” the directories, they delete...
SAS data sets and other files. The %DO loop in lines 50-163 creates SAS, LOG, LST, and RTF files to demonstrate the macros in this paper. Again, to emphasize they are contrived for demonstration purposes. The sections below will discuss their respective macros and calls of them, if necessary.

For simplicity (and some safety), consider the following code to create folders in the WORK directory that should be deleted at the end of an interactive SAS session:

```sas
options dlcreatedir;
libname sdtm "%sysfunc(pathname(work))/sdtm";
libname v_sdtm "%sysfunc(pathname(work))/v_sdtm";
```

**MAC_U_BATCH_SUBMIT**

To introduce this suite of utility and reporting macros with the log check macro might seem reasonable, since one should generally always check one’s log before any other action, this is an appropriate place to start. To be assured that no “artifact”, such as a LIBREF, FILEREF, or global macro variable had the wrong value, each production run of a program should be executed in a fresh SAS session. The log will reflect whether that occurred, but a few lines could slip by even a very astute eye. Batch submission of each program assures a fresh SAS session. Note that developing in PC SAS, i.e., interactive SAS, does not preclude, for example, submitting the program of interest for batch submission within that same session. Appendix 2 presents MAC_U_BATCH_SUBMIT, a SAS macro to batch submit either a single program or a list of programs contained in a data set.

The author adopted the convention of starting the name every macro with “MAC_” so that it is easier to find macro in code, especially when reading every SAS program in a drive or directory. A manager or lead can perform such a task to assist with decisions like when should a macro be considered for promotion from study-level to compound-level or from compound-level to enterprise-level or what would be the impact of updating or retiring a macro, i.e., how widespread is the use of the macro and when was it last used? The second component indicates classification: U-utility, R-reporting, DEBUG-debugging, PGM-defined in the current program, et cetera. The third component is a brief description (what, perhaps, was the name of the macro before adoption of this convention). The author adopts the convention that a debugging macro, for instance, should never appear in a production run, and “mac_debug_” readily identifies such macros. Finally, a warning: when obtaining the files for batch submission, one needs to be sure not to include the batch submission program or, potentially a call to MAC_U_BATCH_SUBMIT (or something similar). Otherwise, one may generate an endless loop: a batch submission program batch submitting itself.

While developing in an interactive session, one can submit snippets of code or the full program, but the final test should be a batch submission to ensure a fresh SAS session. The following “one-liner” in the same interactive session suffices (note an interactive SAS session can have multiple editors):

```sas
%mac_u_batch_submit
(pathfile = C:\Users\vielk\Documents\My SAS Files\9.4\AD221\ad221_batch.sas);
```

Lines 27-32 of Appendix 1 shows one way to create a data set of paths and files names of files that can be used as the input data set for the macro parameter DS (line 2 of Appendix 2). Using Windows Explorer, another way to create a list of path and file names is to select the files of interest and Shift-right click and select “Copy as path” (Figure 1).
One can then paste the results into the DATALINES of a data step for instance:

```
"C:\Users\vielk\Documents\My SAS Files\9.4\sdtm\t_1_3.sas"
"C:\Users\vielk\Documents\My SAS Files\9.4\sdtm\f_1_3.sas"
"C:\Users\vielk\Documents\My SAS Files\9.4\sdtm\l_1_2.sas"
"C:\Users\vielk\Documents\My SAS Files\9.4\sdtm\l_1_3.sas"
"C:\Users\vielk\Documents\My SAS Files\9.4\sdtm\t_1_2.sas"
"C:\Users\vielk\Documents\My SAS Files\9.4\sdtm\t_1_1.sas"
```

Note the order or, rather, lack thereof in the pasted files. If this is the technique used to create a batch submit program, then being aware of the order and changing it is essential. For instance, ADSL.sas should be the first program run, followed by V_ADSL.sas. The same ordering issue is true for the directory read approach to create the list of files.

The macro ultimately generates the command line that is submitted to the operating system via the X statement:

```
1 + x "C:\Program Files\SAS9\SASFoundation\9.4\sas.exe" -CONFIG "C:\Program Files\SAS9\SASFoundation\9.4\sas.cfg" -NOSPLASH -NOSTATUSWIN -SASUSER !userprofile -SYSIN "C:\Users\vielk\Documents\My SAS Files\9.4\PharmaSUG\2023\AD221\ad221_batch.sas"
2 + -NOLOG -ALTLOG "C:\Users\vielk\Documents\My SAS Files\9.4\PharmaSUG\2023\AD221\ad221_batch.log" -PRINT "C:\Users\vielk\Documents\My SAS Files\9.4\PharmaSUG\2023\AD221\ad221_batch.lst" ' ;
```

The macro performs some "housecleaning", like checking if the files and paths exists and generating the LOG and LST paths and files names by substitution of the PATHFILE of the program, the patterns of which can be replaced with a macro parameter with slight programming, if desired. Note the use of -NOLOG and -ALTLOG. The purpose of this is to provide a "shadow" log, assuming that the log will be redirected in the program or macro that initializes the programming environment, including generating the paths and file names of the log and lst and redirecting the log via the PRINTTO procedure. The impetus of this was to be able to close the log, releasing the lock, to perform a log check the results of which would be sent to the lst file, all within the current program execution, i.e., no post-processing required. The macro also provides the opportunity to display the paths as a check before the run to be sure that they are correct. While not, yet, common in clinical trial programming, some analysis programs, such as Bayesian or Mixed models can take hours or longer to run. When genomics is required, data sets that can make LB/ADLB looks small, we might see increased CPU times. Currently, the submissions are in sequence, not parallel, but that can be easily updated for every program at the same hierarchy, i.e., each
program in a given level, those that do not depend on other programs at that level, can be sent to separate nodes or for asynchronous execution via the SAS System Option "NOXSYNC".

**MAC_U_DIRECTORY_READ**

Appendix 3 presents the macro MAC_U_DIRECTORY_READ. This macro reads the directories and files in a path (directory), optionally recursively, and optionally, obtains file information using the SAS FINFO() function (mileage may vary, but bytes and last modified datetimes seem consistently reported in Windows, in the experience of the author). This can be thought of as using the DOS DIR command or the Unix ls command. The data set in Figure 2 is generated by the following macro call:

```sas
%mac_u_directory_read
( path = C:\Users\vielk\Documents\My SAS Files\9.4\PharmaSUG )
```

![Figure 2](image)

Figure 2. The data set created by a call of MAC_U_DIRECTORY_READ.

Examples of calls of MAC_U_DIRECTORY_READ with alternate values of the macro parameters appear below. For instance, a recursive read of a root drive may take hours or only .sas files are of interest (though, hygiene might suggest not to mix file types in certain folders, we find it still occurs).

**MAC_U_LOG_CHECK**

Appendix 4 presents the macro MAC_U_LOG_CHECK. This macro hard codes patterns (phases or words) to search for in (log) files. The macro parameter, PRINT_ALL (Line 4), is a verbose mode, so that the reviewer can see all of the patterns and the number of occurrences, including 0. If a pattern is found, its frequency is reported and one log line with its line number is reported. The data set is deleted by default (Line 8). The macro reports the path, filename, Last Modified datetime stamp, and the number of log lines read:

```sas
%mac_u_log_check
( log_filename = %sysfunc( getoption( SASUSER ))\sdtm\f_1_1.log
 , print_file = %str() )
```

<table>
<thead>
<tr>
<th>Path</th>
<th>File</th>
<th>Last Modified</th>
<th>Records read from the log:</th>
<th>log lines read</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\Users\vielk\Documents\My SAS Files\9.4\sdtm\f_1_1.log</td>
<td>f_1_1.log</td>
<td>27FEB2023:18:51:07</td>
<td>29</td>
<td>29</td>
</tr>
</tbody>
</table>

***** No messages found *****

The Validation programmer may not be able to judge the number of appropriate log lines, but a truncated or empty log will generate the target of zero messages:

```sas
data _null_; file "%sysfunc( getoption( SASUSER ))\sdtm\empty.log" ; put "" ; run ;
```
%mac_u_log_check
{ log_filename = %sysfunc( getoption( SASUSER ))\$dtm\empty.log
, print_all = N
, print_file = %str()
}
C:\Users\vielk\Documents\My SAS Files\9.4\$dtm\empty.log

Path File Last Modified
C:\Users\vielk\Documents\My SAS Files\9.4\$dtm empty.log 19MAR2023:12:33:14

Records read from the log: 1

***** No messages found *****

An example of a log generated by code that MUST be corrected is:

proc printto
    log = "%sysfunc( getoption( SASUSER ))\$dtm\positive_control.log" ;
run ;
data _null_
    x = scan( "Positive control" , y , " ") ;
run ;
proc printto ;
run ;
%mac_u_log_check
{ log_filename = %sysfunc( getoption( SASUSER ))\$dtm\positive_control.log
, print_all = N
, print_file = %str()
}
C:\Users\vielk\Documents\My SAS Files\9.4\$dtm\positive_control.log

Path File Last Modified
C:\Users\vielk\Documents\My SAS Files\9.4\$dtm positive_control.log 19MAR2023:12:44:46

message frequency
-----------------------------------------------
UNINITIALIZED
_ERROR_=1

Records read from the log: 22

Line = 11
NOTE: Variable y is uninitialized.

Line = 13
x= y=. _ERROR_=1 _N_=1

Certainly, the lines above (11 and 13) are not enough to decipher the origins of the issues, but they are also, typically, not enough to violate requirement of 100% independent peer programming. For absolute compliance, the macro can be modified to not display the lines from the peer program’s log.

MAC_U_PATH_LOG_CHECK

Appendix 5 presents the MAC_U_PATH_LOG_CHECK macro. This macro calls MAC_U_LOG_CHECK, creating a “rolling” data set of findings, for presentation or processing. Multiple paths (Line 5) separated by a delimiter (Line 6) can be provided:
The results are summarized by file within a path, by path, and in total:

<table>
<thead>
<tr>
<th>File</th>
<th>Last Modified</th>
<th>Records</th>
<th>Messages</th>
<th>Log Has Message</th>
<th>Log Has Warning</th>
<th>Log Has Error</th>
<th>Log Has Stops</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty.log</td>
<td>19MAR2023:12:33:14</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f_1_1.log</td>
<td>27FEB2023:18:51:07</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>positive_control.log</td>
<td>19MAR2023:12:44:46</td>
<td>22</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t_1_9.log</td>
<td>27FEB2023:18:50:57</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Total in path: 2 1 0 1 0

Logs in path: 32

Path = C:\Users\vielk\Documents\My SAS Files\9.4\v_sdtm

<table>
<thead>
<tr>
<th>File</th>
<th>Last Modified</th>
<th>Records</th>
<th>Messages</th>
<th>Log Has Message</th>
<th>Log Has Warning</th>
<th>Log Has Error</th>
<th>Log Has Stops</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_f_1_1.log</td>
<td>27FEB2023:18:51:13</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>v_t_1_9.log</td>
<td>27FEB2023:18:51:02</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Total in path: 0 0 0 0

Grand total in all paths: 2 1 0 1 0

Logs in path: 30
Total Logs in all paths: 62

Clearly, from the perspective of someone who must audit the files in preparation for a delivery, clean logs (and, as we will see below, clean COMPARE’s) greatly aid the accuracy and efficiency of the review. Even with attestations by the Main and Validation programmers and the biostats, one may want to visit the logs with issues to verify that they are not critical or fatal. Setting the macro parameter PRINT_MESSAGES_GT_0, for instance, will provide details, but the reading the offending log may be required by a neutral (authorized) person (non-peer programmer). A list of specific logs can also be provided to the macro, for instance, by obtaining a list of every SAS file in a directory and deriving the log paths and names from them (substitution, for instance):

```sas
%mac_u_directory_read
  ( path = %sysfunc( getoption( SASUSER ))\sdtm
    , recursive = N
    , finfo = Y
    , code = if prxmatch( "/\sas$/i" , trim( pathfile )) then
  ) ;

data dir_read_logs
  ( keep = log )
  ;
set dir_read ;
log = prxchange( "s/(?<\.)sas/log/\i" , 1 , trim( pathfile ));
run ;
%mac_u_path_log_check
  ( in_ds = dir_read_logs ) ;
```

For brevity, the results are not shown, but these data can be used to check if any SAS file was (inadvertently) updated since the logs were created:
proc sql;
select coalesce(
    scan( a.pathfile, -1, "\" ", 1, "." )
  , scan( b.file
    , 1, "," )
)
  as base
  , length = 50
  , a.last_modified
  , b.last_modified as last_modified_log
from dir_read as a
  full join log_check_all as b
on scan( a.pathfile, -1, "\" ", 1, "." ) = scan( b.file, 1, "," )
having a.last_modified > b.last_modified
order by base;
quit;

<table>
<thead>
<tr>
<th>base</th>
<th>last_modified</th>
<th>last_modified_log</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1_1</td>
<td>27FEB2023:19:00:10</td>
<td>27FEB2023:18:50:15</td>
</tr>
</tbody>
</table>

**MAC_R_COMPARE**

Appendix 6 presents the MAC_R_COMPARE macro. This macro parses any COMPARE output in the lst file provided or in all of the lst files in a directory. For brevity, neither the COMPARE output will be displayed, but using the HELP = Y macro parameter (Line 24) displays an annotated example in the log. The example call below creates the output in Figure 3:

```sas
%mac_r_compare_report
  (path = %sysfunc(pathname(v_sdtm)), file = *.lst);
```

Figure 3. The results of MAC_R_COMPARE_REPORT.

MAC_U_COMPARE_REPORT generates a global macro variable (Line 23) that indicates whether any issue was found in any COMPARE RESULTS of the given lst file. This macro variable can be used in a batch submission program to end it without further batch submitting any other program. For instance, if ADSL does not “pass” Validation by virtue of having a different number of observations, variables, or having mismatching values, the processing depending on ADSL, including every subsequent ADaM data set and TFL, should be stopped. SAS can email, if IT allows certain things, so the team can be alerted as soon as fatal issue is encountered.

Allowing issues in a single log may expedite programming at that moment, but it creates problems for audits at the delivery level. For instance, a Validation programmer may reason that her or his program and data sets are not part of the delivery so failing to drop certain “transactional” variables may be acceptable, but when viewing the report in Figure 3, one cannot tell if the Main data set omitted...
required variables. Further, variables (and observations) not common to both data sets are not
COMPARE’d and one should not assume that they are correct or not required. The macro creates a data
set that can be archived or used for metrics a manager or stakeholder might use to judge the quality and
efficiency of programming to create more accurate budgets and timelines or to allocate resources
(programmers) differently.

MAC_U_DELETE

Appendix 7 presents the MAC_U_DELETE macro. The essence of this macro is the DATASETS
procedure and its DELETE statement. The macro checks for the existence of the file before deleting it to
avoid messages in the log when it may not exist. The DS macro parameter (Line 5) can take a mixed list
of one- or two-level SAS data sets; the data sets can be in different libraries (directories). Several macros
call this macro. This is an “uncontrolled” delete, no prompt occurs. In certain environments, the author
may use a PW= data set option to avoid inadvertent or unintended deletions (or changes), but that is not
a common approach in clinical trial programming.

DEMONSTRATIONS USING %MAC_PGM_TFL

A few contrived examples of “mismatch” can be generated, by design, with the %MAC_PGM_TFL macro.
This is just one way to change the files. Positive and negative control test are usually a wise addition to
DUT (Device Under Testing).

TEST_1

The first demonstration of an example of these programs used shows how to end batch submission if
Validation fails and the abbreviated results of the MAC_RCOMPARE_REPORT macro:

    %mac_prg_tfl
    ( test_1      = drop = col_1 ) ;

The log, among other things, will show:

    ALERT:  v_t_1_1.lst has mismatches

The dropping of COL_1 not only exits MAC_PGM_TFL prematurely, but also shows in the summary of
the compare results:

    %mac_r_compare_report
    ( path = %sysfunc( pathname( v_sdtm ))
            , file = *.lst
    ) ;

<table>
<thead>
<tr>
<th>Obs</th>
<th>file</th>
<th>data</th>
<th>comp</th>
<th>issue</th>
<th>vars</th>
<th>common</th>
<th>issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>v_t_1_1.lst</td>
<td>SDTM.T_1_1</td>
<td>V_SDTM.V_T_1_1</td>
<td>7</td>
<td>6</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

COMPARE results with matching variable and observations numbers and no unequal values

--------

0
The criteria (Lines 396-399) for the concluding statement can be updated to suite the needs of the team or satisfy the requirements set by the SOPs/Wis. Using a macro parameter might be a good option. Lines 411-417 define criteria to populate the macro parameter MV_FAIL (Line 23, default value COMPARE_FAIL).

**TEST_2**

TEST_2 demonstrates the situation in which the number of observations differ. When using the ID statement for the COMPARE procedure, the number of observations in both data sets can match, but they may not match on the ID variable. The LISTALL option to the COMPARE statement is useful.

```sas
%tfl
  ( test_2      = obs = 1 ) ;
%mac_r_compare_report
  ( path = %sysfunc( pathname( v_sdtm ))
    , file = *.lst
  ) ;

obs_in_ obs_ obs_w_ obs_all_
file        data         comp       data_obs   common  issue    unequal      equal
v_t_1_1.lst SDTM.T_1_1 V_SDTM.V_T_1_1 8       1       Y 0           1
=======
0

Note that although these two data sets have one observation in common, the value of variables match and OBS_W_UNEQUAL is 0 (zero).

**TEST_3**

TEST_3 demonstrates mismatching (discrepant values). Although this is the final demonstration test, matching values (accuracy) is the primary criterion of Validation. In this type of programming, the values have to match exactly, including formatting, such as leading spaces or SAS inline styles, potentially complicating Validation. Sometimes, additional items cannot be avoided, such as superscripts. Leading spaces in SAS, for instance, to indent a row label under its section header label, is avoidable in SAS using a COMPUTE block in the REPORT procedure and the STYLE INDENT. The author demonstrates this and how to quickly display mismatches that are too long to be displayed by COMPARE results in another paper in this conference.

```sas
%tfl
  ( test_3      = call missing( col_1 ) %str();)
%mac_r_compare_report
  ( path = %sysfunc( pathname( v_sdtm ))
    , file = *.lst
  ) ;

Obs       file        data         comp       data_obs   common  issue    unequal      equal
1         v_t_1_1.lst SDTM.T_1_1 V_SDTM.V_T_1_1 10       0
=======
10

**CONCLUSION**

This paper contributes to the essential process of Validation in clinical trial programming by providing programmatic approaches to audit an entire delivery (which may comprise multiple directories). Not only
do we owe a duty to the subjects who volunteer to enter our trial, potentially with meaningful risk, and their loved ones and healthcare providers, to be certain of our work, we program in one of the most regulated fields on trials that can cost millions of dollars and last years, indeed the cumulative costs can run into the tens of millions of dollars and beyond. A second programmatic approach that executes in minutes, both displays the results concisely and creates data sets of them, and itself leaves an audit trail should be a welcomed addition to, but not a substitution for any manual review.

The macros provided can be used by individual programs in production run. For instance, a log check can be incorporated into a macro used at the very end of a program (to close and clean up, for instance). The author creates a batch_submission.sas file for most folders in which he works. In addition to a call to MAC_U_BATCH_SUBMIT, even while developing, the author then calls MAC_U_LOG_CHECK, followed by two calls to MAC_RCOMPARE_REPORT when Validating. The call to the MAC_U_LOG_CHECK in this situation display the results in the output window of interactive SAS. The first call to MAC_RCOMPARE_RESULTS examines the result from the files only in the batch_submission.sas and the second reports the COMPARE results for the entire directory. This assures the author that the status, for instance, of his current ADaM program, such as V_ADAE.sas, is not affected by issues in the COMPARE results of V_ADSL.sas, on which it relies.

Programming in clinical trials can have strict deadlines and strict rules. In the case of a delivery in a set number of business days after a data lock, the time to just review the Validation results may be nearly prohibitive: viewing the COMPARE results of just 30-50 ADaM data sets could take hours; TFL, which can easily require 300-500 programs, have the added complexity that the output must be viewed for formatting issues, even if the VALDS match 100%, and their titles and footnotes also have to be verified. Programmatically approaching some issues, like batch submission in appropriate order and summarizing logs and COMPARE results at the delivery level may increase the accuracy, completeness of the review, and efficiency, but the SOPs/WIs must allow such approaches. The ability to audit directories over time can also provide metrics to a manager that indicate which programs might cause issues over several deliverables or how to create realistic timelines or allocate resources and budgets. The ability to view a log from a batch submission program and the datetime stamps to assure a manager or lead that the programs were run correctly and that the files have appropriate (relative) ages increases confidence in the quality of the delivery.

The macros and code in this paper are provided "as is". The author and his employers assume no responsibility for their use, but do appreciate notification of errors, bugs, corrections, or suggestions.

REFERENCES


CONTACT INFORMATION

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

Kevin R. Viel, Ph.D.

Navitas Data Sciences
kevin.viel@navitaslifesciences.com
www.navitasdatasciences.com

Histonis, Incorporated
kviel@histonis.org

Any brand and product names are trademarks of their respective companies.
Appendix 1. A suggested macro to provide test programs, logs, lsts, and output.

```plaintext
%macro mac_pgm_tfl(
    list = t f l,
    number = 10,
    end_on_fail = Y,
    test_1 = %str( ),
    test_2 = %str( ),
    test_3 = %str( )
);

%mac_u_directory_read(
    path = %sysfunc( pathname( sdtm )),
    recursive = N,
    code = if prxmatch( "/(?:lst|log|sas|rtf$/i" , trim( pathfile )) then
        %do %while ( &l. ne %str( ) )
            %do __j = 1 %to &number. ;
            proc printto log = "%sysfunc( pathname( sdtm ))\&l._1_&__j..log" new;
            data sdtm.&l._1_&__j. ;
            file "%sysfunc( pathname( sdtm ))\&l._1_&__j..sas" ;
            put "placeholder" ;
            length page_1 $ 1
            page_order_1 8
            section_1 $ 1
            section_order_1 8
            row_1 $ 6
            row_order_1 8
            col_1 $ 10
            ;
            retain page_1 "Safety Population"
            page_order_1 1
            section_1 "Sex n(\%"
            section_order_1 1
            ;
            do row_order_1 = 1 to call( ranuni( 0 ) * 10 ) ;
            row_1 = cat( "Row ", strip( put( row_order_1 , 8. )) ) ;
            col_1 = cat( strip( put( row_order_1 , 8. )) )
        %end;
    %end;

%end;
```
13

13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170

; "1"
; strip( put( row_order_1 / 50 * 100 , 8.1 ))
; "1"
output ;
end ;
run ;
ods rtf
file = "%sysfunc( pathname( sdtm ))\%sdtm.\%sdtm.rtf" /
proc print
data = sdtm.\sdtm.\.
run ;
ods rtf close ;
proc printto ;
run ;
/*******
data null ;
call sleep( 5 , 1 ) ;
run ;
proc printto
log = "%sysfunc( pathname( v_sdtm ))\v_sdtm.\v_sdtm.log"
print = "%sysfunc( pathname( v_sdtm ))\v_sdtm.\v_sdtm.lst"
new
run ;
data v_sdtm.v_sdtm.\.
( \&test_1. )
file "%sysfunc( pathname( v_sdtm ))\v_sdtm.\v_sdtm.sas" ;
put "placeholder" ;
set sdtm.\sdtm.\.
( \&test_2. )
( \&test_3. )
run ;
/*******
proc compare
data = sdtm.\sdtm.\.
comp = v_sdtm.v_sdtm.\.
listall
run ;
proc printto ;
run ;
%mac_r_compare_report
{ path = %sysfunc( pathname( v_sdtm ))
, file = v_sdtm.\v_sdtm.\.
, print = N
, list = W
};
%if &compare_fail. = 1
%then
%do;
%put A%str(LERT: v_sdtm.\v_sdtm.\.
has mismatches ;
%if &end_on_fail. = Y %then %goto __END ;
%end;
%end;
%let __i  = %eval( &__i. + 1 ) ;
%let l    = %sysfunc( scan( &list. , &__i. , %str( ))) ;
%end;
%__END:
ods listing ;
ods results ;
%end
mac_pgm_tfl ;
Appendix 2. The MAC_U_BATCH_SUBMIT macro.

```sas
%macro mac_u_batch_submit
( do = %str(),
  pathfile = %str(),
  derive_log_lst = Y,
  log = %str(),
  lst = %str(),
  sas_exe = \SASHome\SASFoundation\9.4\sas.exe,
  sas_cfg = \SASHome\SASFoundation\9.4\nls\u\sasv9.cfg,
  sasuser = !userprofile,
  nosplash = Y,
  help = YE
)
%do ;
  %let mprint = %sysfunc(getoption(mprint)) ;
  %options mprint = %mprint ;
  %put ___________________________________________________________________________
  %put Description
  %put Macro Parameter
  %put Purpose of program: This utility macro submits SAS programs in the input data set or a single program provided by
  %put Each program is run by
  %put parallel.
  %put %str(Provide only one of DS or PATHFILE.)
  %put %str(End of help)
  %end ;
%macro_end ;

%let delete_ds = N ;
%options noxwait
%let xsync = Y

%if &ds. ne %str() ;
  %if %nrbquote(&pathfile.) ne %str() ;
  %if &ds. = %str() ;
    %let delete_ds = N ;
  %end ;
  %if &help. = Y ;
    %let xmin = %sysfunc(getoption(xmin)) ;
    %let xsync = %sysfunc(getoption(xsync)) ;
    %let xwait = %sysfunc(getoption(xwait)) ;
    %let delete_ds = N ;
  %end ;
%then ;
  %do ;
    %gong __END__ ;
    %put WARNING: LOG and LST must both have values if PATHFILE is populated and DERIVE_LOG_LST = Y ;
    %goto __END__ ;
  %end ;
%end ;
%if &ds. ne %str() ;
  %if &pathfile. = %str() ;
    %let derive_ds = N ;
  %end ;
  %if &pathfile. ne %str() ;
    %let derive_ds = N ;
  %end ;
  %if &log. ne %str() ;
    %let derive_log_lst = Y ;
  %end ;
  %if &derive_log_lst. = Y ;
    %let derive_log_lst = Y ;
  %end ;
  %if &help. = Y ;
    %let xmin = %sysfunc(getoption(xmin)) ;
    %let xsync = %sysfunc(getoption(xsync)) ;
    %let xwait = %sysfunc(getoption(xwait)) ;
    %let derive_log_lst = Y ;
    %let derive_log_lst = Y ;
  %end ;
%end ;
```

and %nrbquote( &pathfile. ) ne %str();
%then
  %do;
    data &ds. ;
    length pathfile log lst $ 1000;
    pathfile = "&pathfile."
      if %nrbquote( &derive_log_lst. ) ne Y then
        log = &log. ;
        lst = &lst. ;
      else
        log = prxchange( "s/\..sas$/\..log/i", 1, trim( prxchange( "s/\..\Pgm/\..log/i", 1, pathfile ) ) ) ;
        lst = prxchange( "s/\..sas$/\..lst/i", 1, trim( prxchange( "s/\..\Pgm/\..lst/i", 1, pathfile ) ) ) ;
     %end;
  %end;
run ;
%let ds = __ds ;
%let delete_ds = Y ;
%end ;
/* Verify that the input data set exists, that the files it contains exists, and that they are .sas files.
  The macro stops once one of these criteria fail.
*/
%if %sysfunc( exist( &ds. )) then
  %let __flag = 0 ;
  data _null_ ;
  set &ds. ;
  __fileexist = fileexist( pathfile ) ;
  __sas = prxmatch( "s/\..sas$/\..sas$/i", strip( pathfile ) ) ;
  if __fileexist = 0 or __sas = 0 then
    if __fileexist = 0 then put "ER" "ROR: the file " pathfile "does not exist. Processing will stop" ;
    if __sas = 0 then put "ER" "ROR: the file " pathfile "is not a .sas program. Processing will stop" ;
    call symput( "__flag" , "1" ) ;
    stop ;
  end ;
  /* Check for the existence of log and lst paths */
  logpath = prxchange( "s/\..sas$/.log/i", 1, strip( log ) ) ;
  __logexist = fileexist( logpath ) ;
  if __logexist = 0 then put "ER" "ROR: the file " logpath "does not exist. Processing will stop" ;
  call symput( "__flag" , "1" ) ;
  stop ;
end ;
/* Check for the existence of log and lst paths */
lstpath = prxchange( "s/\..\..\\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..

then
  do;
  put "ER" "ROR: the path " lstpath "does not exist. Processing will stop" ;
  call symput( "_flag" , "1" ) ;
  stop;
end;
run;
if _flag. = 1 then goto __END ;
end;

%if &__flag. = 1 %then %goto __END ;
%end ;
%else
  %do ;
  %put ER%str(ROR:) the data set &ds. does not exist. ;
  %goto __END ;
  %end ;
data _null_ ;
set &ds. ;
if &display_paths. ne Y
  %then
    %do ;
    call execute( cat( "x '" , %unquote(%str(%" %")&sas_exe.%str(%" %") )
      %if &nosplash.    = Y %then , "-NOSPLASH " ;
      %if &nostatuswin. = Y %then , "-NOSTATUSWIN " ;
      %if %nrbquote(&sasuser.) ne %str() %then , "-SASUSER &sasuser. " ;
      , "-SYSIN "
      , "," ) ;
    %end ;
  %else
    %do ;
    length file_path file_log file_lst $ 100 ;
    file_path = scan( pathfile , -1 , "\" ) ;
    file_log  = scan( log      , -1 , "\" ) ;
    file_lst  = scan( lst      , -1 , "\" ) ;
    put $1 file_path
      $100 pathfile
        $1 file_log
          $100 log
            $1 file_lst
              $100 lst
        ;
    %end ;
  %end ;
%__END: 
options &xwait.
&xsync.
&smin.

%end mac_u_batch_submit ;
Appendix 3. The MAC_U_DIRECTORY_READ macro.

1 macro mac_u_directory_read
2 { path =
3   , path_delim = #
4   , keep     = pathfile
5   , recursive = Y
6   , levels = 20
7   , output_dir = Y
8   , output_files = Y
9   , out = dir_read
10  , code = $str()%
11  , os_path_delim = \
12  , info = N
13  , foptname = Last Modified
14  , foptname_delim = #
15  , help = N
16  } ;
17
18 %if help = Y
19 then
20 %do ;
21   %let mprint_orig = %sysfunc(getoption(mprint));
22   options nomprint ;
23   skip ;
24   skip ;
25   %put
26   %put Purpose of program: This utility macro ;
27   %put %str(1) Reads the contents of a list of directories (paths). ;
28   %put %str(2) Optionally recursively reads the child directories. ;
29   %put %str(3) Optionally limits the level of recursion. ;
30   %put %str(4) Optionally includes free-text code prior to the OUTPUT statement. ;
31   %put %str(5) Optionally provides FINFO. ;
32   %put %str(6) Optionally limites the level of recursion. ;
33   %put %str(7) Reads
34   %put Macro Parameter Description ;
35   %put %str(8) Optionally includes free
36   %put %str(9) Optionallly limites the level of recursion. ;
37   %put %str(10) Reads
38   %put %str(11) Optionallly limites the level of recursion. ;
39   %put %str(12) Optionallly limites the level of recursion. ;
40   %put %str(13) Optionallly limites the level of recursion. ;
41   %put %str(14) Optionallly limites the level of recursion. ;
42   %put %str(15) Optionallly limites the level of recursion. ;
43   %put %str(16) Optionallly limites the level of recursion. ;
44   %put %str(17) Optionallly limites the level of recursion. ;
45   %put %str(18) Optionallly limites the level of recursion. ;
46   %put %str(19) Optionallly limites the level of recursion. ;
47   %put %str(20) Optionallly limites the level of recursion. ;
48   %put %str(21) Optionallly limites the level of recursion. ;
49   %put %str(22) Optionallly limites the level of recursion. ;
50   %put %str(23) Optionallly limites the level of recursion. ;
51   %put %str(24) Optionallly limites the level of recursion. ;
52   %put %str(25) Optionallly limites the level of recursion. ;
53   %put %str(26) Optionallly limites the level of recursion. ;
54   %put %str(27) Optionallly limites the level of recursion. ;
55   %put %str(28) Optionallly limites the level of recursion. ;
56   %put %str(29) Optionallly limites the level of recursion. ;
57   %put %str(30) Optionallly limites the level of recursion. ;
58   %put %str(31) Optionallly limites the level of recursion. ;
59   %put %str(32) Optionallly limites the level of recursion. ;
60   %put %str(33) Optionallly limites the level of recursion. ;
61   %put %str(34) Optionallly limites the level of recursion. ;
62   %put %str(35) Optionallly limites the level of recursion. ;
63   %put %str(36) Optionallly limites the level of recursion. ;
64   %put %str(37) Optionallly limites the level of recursion. ;
65   %put %str(38) Optionallly limites the level of recursion. ;
66   %put %str(39) Optionallly limites the level of recursion. ;
67   %put %str(40) Optionallly limites the level of recursion. ;
68   %put
69   %put End of help
70 options mprint_orig. ;
71 %goto __END ;
72 %end ;
73
74 %if output_dir. ne Y
75 and output_files. ne Y
76 %then
77 %do ;
78   %put ER%str(ROR:) OUTPUT_DIR = &output_dir. ;
79   %put ER%str(ROR:) OUTPUT_FILES = &output_files. ;
80   %goto __END ;
81 %end ;
82 %mac_u_delete
83 { ds = &out.
84  dir
85  } ;
86 %data &out.
87  keep = %keep.
88 } ;
%if %sysfunc(prxmatch(/pathfile/ , &keep.)) = 0 %then pathfile ;
%do;
level
%if %sysfunc(prxmatch(/Filename/i, &foptname.)) %then filename ;
%else %if %sysfunc(prxmatch(/RECFM/i, &foptname.)) %then recfm ;
%else %if %sysfunc(prxmatch(/LRECL/i, &foptname.)) %then lrecl ;
%else %if %sysfunc(prxmatch(/File Size (bytes)/i, &foptname.)) %then file_size ;
%else %if %sysfunc(prxmatch(/Last Modified/i, &foptname.)) %then last_modified ;
%else %if %sysfunc(prxmatch(/Create Time/i, &foptname.)) %then create_time ;
%end;

length __path   $ 512
object   $ 256
pathfile $ 512
%if %sysfunc(prxmatch(/\bfile/ , &keep. )) %then file $ 256 ;
%do;
foptname $ 100
finfo    $ 512
%if %sysfunc(prxmatch(/Filename/i, &foptname. )) %then filename $ 512 ;
%else %if %sysfunc(prxmatch(/Owner Name/i, &foptname. )) %then owner $  30 ;
%else %if %sysfunc(prxmatch(/Group Name/i, &foptname. )) %then group $  50 ;
%else %if %sysfunc(prxmatch(/Access Permission/i, &foptname. )) %then access_permission $  30 ;
%else %if %sysfunc(prxmatch(/Last modified/i, &foptname. )) %then last_modified ;
%else %if %sysfunc(prxmatch(/File Size (bytes)/i, &foptname. )) %then file_size_bytes ;
%end;

do __p = 1 to countc("&path.", "&path_delim.") + 1 ;
  /* Remove the trailing OS path delimiter, if it is present */
  __path   = prxchange("s/(.*)(?:\&os_path_delim.)$/$1/", 1,
                     strip(scan("&path.", __p, "&path_delim.")));
  __level  = 1 ;
  __rc    = filename("dir", __path ) ;
  dopen   = dopen("dir") ;
  /**************/
  if __rc > 0 then do ;
    dnum = dnum( dopen ) ;
    if dnum > 0 then do ;
      do objectnum = 1 to dnum ;
        object   = dread( dopen , objectnum ) ;
        pathfile = catx("&os_path_delim.", __path,
                      object ,
                      ) ;
        call missing(finfo
                   %if %sysfunc(prxmatch(/Filename/i, &foptname. )) %then , filename ;
                   %else %if %sysfunc(prxmatch(/Owner Name/i, &foptname. )) %then , owner ;
                   %else %if %sysfunc(prxmatch(/Group Name/i, &foptname. )) %then , group ;
                   %else %if %sysfunc(prxmatch(/Access Permission/i, &foptname. )) %then , access_permission ;
                   %else %if %sysfunc(prxmatch(/Last modified/i, &foptname. )) %then , last_modified ;
                   %else %if %sysfunc(prxmatch(/File Size (bytes)/i, &foptname. )) %then , file_size_bytes ;
                   ) ;
  /* Attempt to open the object as a directory */
  ___rc = filename("dirchild", pathfile ) ;
  ___dopen = dopen("dirchild") ;
  ___rc = dclose(___dopen) ;
  ___rc = filename("dirchild") ;
  %if %sysfunc(prxmatch(/\bpath/ , &keep. )) %then %do ;
    if ___rc = 0 then do ;
      path = __path ;
      %if %sysfunc(prxmatch(/\bfile/ , &keep. )) %then file = object %str(;) ;
    %end ;
  %end ;
  %end ;
%end ;

/*****/
185 end;
186 size
do;
188 path = pathfile;
189 endif $sysfunc( prxmatch( "\bfile\b" , &keep. )) $then file = " " $str(;) ;
190 end;
191 if $finfo. = Y
192 then
193 %do;
194 rc = filename( "fn" , pathfile ) ;
195 fid = fopen( "fn" ) ;
196 if fid ne 0
197 then do ;
198 call missing( finfo
199 %if $sysfunc( prxmatch( /Filename/i , &foptname. )) %then , filename ;
200 %if $sysfunc( prxmatch( /Owner Name/i , &foptname. )) %then , owner ;
201 %if $sysfunc( prxmatch( /Group Name/i , &foptname. )) %then , group ;
202 %if $sysfunc( prxmatch( /Access Permission/i , &foptname. )) %then , access_permission ;
203 %if $sysfunc( prxmatch( /Last Modified/i , &foptname. )) %then , last_modified ;
204 %if $sysfunc( prxmatch( /File Size (bytes)/i , &foptname. )) %then , file_size_bytes ;
205 %end ;
206 else put "WARN: "OfWork: " foptname= finfo= ;
207 end;
208 %end ; /* END OF &finfo. = Y */
209 %if $output_files. = N
210 then
211 %do;
212 if __dopen > 0
213 then do ;
214 &code.
215 output ;
216 %end;
217 %else
218 %do;
219 &code.
220 output ;
221 %end;
222 %end ; /* CYCLED THROUGH objectnum = 1 to dnum */
223 %end ; /* END OF dopen > 0 */
224 __rc = dclose( dopen ) ;
225 __rc = filename( "dir" ) ;
226 stop ;
227 %if $finfo. = Y
228 and $sysfunc( prxmatch( /Last Modified/i , &foptname. ))
229 $then format last_modified datetime20. $str(;) ;
run;

%if $recursive. = Y
then
  do;
    proc sql noprint;
      select count( * ) into : dirs
      from &out.
      where __dopen > 0
    ;
    quit;
    %if &sqlobs. > 0
    then
      %do;
        data __dirs
        ( keep = __path __level
          set &out.
          ( rename = ( pathfile = __path )
            where = ( __dopen > 0 )
          )
        );
      %end;
    %if &output_dir. = N
    then
      %do;
        data &out.
        set &out.
        ( where = ( __dopen = 0 )
        );
      %end;
    %let level = 1;
    %do %while (     &dirs.  > 0
       and &level. < &levels.
    )
      %let level = %eval( &level. + 1 )
      data __dirs
      ( keep = __path __dopen __level
        __level = __level + 1;
        __rc   = filename( "dir" , __path )
        dopen   = dopen( "dir" )
      );
      length __path   $ 512
      object   $ 256
      pathfile $ 512
      %if %sysfunc( prxmatch( /pathfile/ , &keep. )) = 0
       then pathfile ;
      __dopen = 0
      %if &finfo. = Y
      then
        %do;
          %if %sysfunc( prxmatch( /Filename/i , &foptname. )) = 0
           then filename   $ 512
          %if %sysfunc( prxmatch( /Owner Name/i , &foptname. )) = 0
           then owner   $  30
          %if %sysfunc( prxmatch( /Group Name/i , &foptname. )) = 0
           then goup   $  50
          %if %sysfunc( prxmatch( /Access Permission/i , &foptname. )) = 0
           then access_permission $  30
          %if %sysfunc( prxmatch( /Last modified/i , &foptname. )) = 0
           then last_modified $  30
          %if %sysfunc( prxmatch( /File Size (bytes)/i , &foptname. )) = 0
           then file_size_bytes $ 512
        %end;
      %end;
    %end;
  %end;
%end;

%let os_path_delim = /;
if dopen > 0
then
  dnum = dnum( dopen ) ;
if dnum > 0
then
  do ;
  do objectnum = 1 to dnum ;
  object   = dread( dopen , objectnum ) ;
  pathfile = catx( "&os_path_delim." , __path , object ) ;
  call missing( finfo )
  %if %sysfunc( prxmatch( /Filename/i , &foptname. )) %then , filename = finfo ;
  %if %sysfunc( prxmatch( /Owner Name/i , &foptname. )) %then , owner = finfo ;
  %if %sysfunc( prxmatch( /Group Name/i , &foptname. )) %then , group = finfo ;
  %if %sysfunc( prxmatch( /Access Permission/i , &foptname. )) %then , access_permission = finfo ;
  %if %sysfunc( prxmatch( /Last modified/i , &foptname. )) %then , last_modified = input( finfo , datetime20. ) ;
  %if %sysfunc( prxmatch( /File Size (bytes)/i , &foptname. )) %then , file_size_bytes = input( finfo , best. ) ;
  %if %sysfunc( prxmatch( /path/ , &keep. )) %then
    %do ;
    if __dopen = 0
      then
        path = __path ;
        %if %sysfunc( prxmatch( /bfile/ , &keep. )) then file = object %str(;) ;
      end ;
    else
      path = pathfile ;
      %if %sysfunc( prxmatch( /bfile/ , &keep. )) then file = " " %str(;) ;
    end ;
  %end ;
  %if &finfo. = Y
  %then
    %do ;
    rc  = filename( "fn" , pathfile ) ;
    fid = fopen( "fn" ) ;
    if fid ne 0
      then
        do ;
        call missing( finfo )
        %if %sysfunc( prxmatch( /Filename/i , &foptname. )) %then , filename = finfo ;
        %if %sysfunc( prxmatch( /Owner Name/i , &foptname. )) %then , owner = finfo ;
        %if %sysfunc( prxmatch( /Group Name/i , &foptname. )) %then , group = finfo ;
        %if %sysfunc( prxmatch( /Access Permission/i , &foptname. )) %then , access_permission = finfo ;
        %if %sysfunc( prxmatch( /Last modified/i , &foptname. )) %then , last_modified = input( finfo , datetime20. ) ;
        %if %sysfunc( prxmatch( /File Size (bytes)/i , &foptname. )) %then , file_size_bytes = input( finfo , best. ) ;
        %when ( upcase( foptname ) = "FILENAME" ) filename = finfo ;
        %when ( upcase( foptname ) = "OWNER NAME" ) owner = finfo ;
        %when ( upcase( foptname ) = "GROUP NAME" ) group = finfo ;
        %when ( upcase( foptname ) = "ACCESS PERMISSION" ) access_permission = finfo ;
        %when ( upcase( foptname ) = "LAST MODIFIED" ) last_modified = input( finfo , datetime20. ) ;
        %when ( upcase( foptname ) = "FILE SIZE (BYTES)" ) file_size_bytes = input( finfo , best. ) ;
        %otherwise put "WARING: " foptname= finfo= ;
        end ;
      end ;
    close = fclose( fid ) ;
end /* END OF fid ne 0 */
rc = filename( "fn" ) ;
%end ;
%if &output_files. = N
%then
  %do ;
    if __dopen > 0
    then
      do ;
        &code.
        output ;
        end ;
    %end ;
  %else
    %do ;
      &code.
      output ;
      %end ;
end ; /* CYCLED THROUGH objectnum = 1 to dnum */
end /* END OF dnum > 0 */
end /* END OF dopen > 0 */
__rc = dclose( dopen ) ;
__rc = filename( "dir" ) ;
run ;
proc append
  base = &out.
  data = __dirs
  %if &output_dir. = N %then ( where = ( __dopen = 0 ) ) ;
  ;
  run ;
  proc sql noprint ;
  select count( * ) into : dirs
  from __dirs
  where __dopen > 0
  ;
  quit ;
  %if &sqlobs. > 0 %then
    %do ;
      data __dirs
      { keep = __path
      __level
      };
      set __dirs
      { rename = ( pathfile = __path )
      where = ( __dopen > 0 )
      };
      run ;
    %end ;
  %end ; /* END OF ( &dirs. > 0 and &level. < &levels. ) */
%mac_u_delete
  ( ds = __dirs ) ;
%end ; /* END OF recursive = Y */
%else %if &recursive. ne Y
  and &output_dir. = N
%then
    %do ;
      data &out.
      ( where = ( __dopen = 0 ) )
      ;
      run ;
      %end ;
    %if %sysfunc( prxmatch( /pathfile/ , &keep. ) ) = 0
      %then
    %do ;
      proc sql ;
      alter table &out.
      drop pathfile
      ;
      quit ;
    %end ;
    %__END:
%end ; /* _END: */
%end mac_u_directory_read ;
The MAC_U_LOG_CHECK macro.

The MAC_U_LOG_CHECK macro should be run at the end of a program or sub-program to review log file contents. It reports the following:

**Log File Name**
- Pathname of the SAS log file to review

**Print All**
- Print the entire list of messages even those count = 0

**Print Log Filename**
- Pathname of the log file to write the results via PRINTO procedure

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro

**Log Filename**
- Pathname of the SAS log file to review

**Print Append**
- If writing to PRINT_FILE, should the macro append or write a new file?

**Warn Unbalance Quote**
- Whether to include the warning if the quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks.

**Delete**
- 1 results in the deletion of the temporary datasets used or generated by the macro
/* Conditional processing based on whether print_file is a SAS fileref or not. */
%if %nrbquote(&print_file.) ne %str()
  and %sysfunc(prxmatch(/Y|YES/i, &print.))
%then
  %do;
    proc printto
      print = &print_file.;
      if &print_append. = N then New;
    run;
  %end;
/* / */
%if %sysfunc(prxmatch(/Y|YES/i, &print.)) ne 0
%then
  %do;
    %if %sysfunc(prxmatch(/Y|YES/i, &print.)) = 0 %then
      %do;
      data _null_
        file print;
      put "log_filename.";
      run;
      %end;
    %end;
  %end;
/** Last modification date **/
%mac_u_finfo
  (in_ds    = %str(), pathfile = &log_filename.,
   delete   = 0, put_header           = @1   "Path",
   put_variables        = @1   path
                      @150 "File",
                      @200 "Last Modified",
   put_variables        = @1   path
                      @150 file
                      @200 last_modified)
%if &delete. = 1
  %then
    %do;
      proc datasets
        library = WORK nolist;
      delete finfo;
      quit;
    %end;
  %end; /* log_filename is a path/log, not a fileref */
%else %if %sysfunc(prxmatch(/Y|YES/i, &print.))
  %then
    %do;
      data null_ ;
      file print;
      put "log_filename.";
      run;
    %end;
  %end;
  %end; /* print_file is a SAS fileref */
/* ** Last modification date **/
%mac_u_finfo
  (in_ds    = %str(), pathfile = &log_filename.,
   delete   = 0, put_header           = @1   "Path",
   put_variables        = @1   path
                      @150 "File",
                      @200 "Last Modified"
   put_variables        = @1   path
                      @150 file
                      @200 last_modified)
%if %sysfunc(prxmatch("/SAS SYSTEM STOPPED PROCESSING THIS STEP/i", line))
then
  do; 
    message = "SAS SYSTEM STOPPED PROCESSING THIS STEP" ;
    log_line = _n_; 
    output __log_messages ;
  goto __end ;
  end ;
%if %sysfunc(prxmatch("/ERROR:/", line))
then
  do; 
    message = "ERROR:" ;
    log_line = _n_; 
    output __log_messages ;
  goto __end ;
  end ;
%end; if prxmatch("/SAS SYSTEM STOPPED PROCESSING THIS STEP/i", line)
%then
  do;
    message = "SAS SYSTEM STOPPED PROCESSING THIS STEP" ;
    log_line = _n_; 
    output __log_messages ;
  goto __end ;
  end ;
%end; if prxmatch("/ERROR:/", line)
%then
  do;
    message = "ERROR:" ;
    log_line = _n_; 
    output __log_messages ;
  goto __end ;
  end ;
%end;
output __log_messages;
goto __end;
end;
if prxmatch("/ERROR \d+-\d+/", line) then
do;
  message = "ERROR \d+-\d+";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
if prxmatch("/_ERROR_=1/", line) then
do;
  message = "_ERROR_=1";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
if prxmatch("/WARNING/", line) and prxmatch("/UWARNING/", line) = 0 and prxmatch("/WARNING: Engine XPORT does not support SORTEDBY operations. SORTEDBY information cannot be copied./", line) = 0 and prxmatch("/WARNING: Some character data was lost during transcoding in column/", line) = 0 /* WARNING: The quoted string currently being processed has become more than 262 characters long. 
You might have unbalanced quotation marks. */
and prxmatch("/WARNING: The quoted string currently being processed has become more than 262 characters long/", line) = 0 then and prxmatch("/WARNING: The quoted string currently being processed has become more than 262 characters long/", line) = 0 then then do;
  message = "WARNING";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
if prxmatch("/ABNORMALLY TERMINATED/i", line) then
do;
  message = "ABNORMALLY TERMINATED";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
if prxmatch("/ALREADY EXISTS/i", line) then
do;
  message = "ALREADY EXISTS";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
if prxmatch("/ARGUMENT TO FUNCTION/i", line) then
do;
  message = "ARGUMENT TO FUNCTION";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
if prxmatch("/COULD NOT BE WRITTEN/i", line) then
do;
  message = "COULD NOT BE WRITTEN";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
if prxmatch("/DIVISION BY ZERO DETECTED/i", line) then
do;
  message = "DIVISION BY ZERO DETECTED";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
if prxmatch("/DOES NOT EXIST/i", line) then
do;
  message = "DOES NOT EXIST";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
if prxmatch("/ERROR \d+-\d+/", line) then
do;
  message = "ERROR \d+-\d+";
  log_line = _n_;
  output __log_messages;
goto __end;
end;
end;

if prxmatch("/ENDSAS/i", line)
then
do;
    message = "ENDSAS";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/EXPERIMENTAL IN RELEASE/i", line)
then
do;
    message = "EXPERIMENTAL IN RELEASE";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/FORMAT WAS TOO SMALL FOR THE NUMBER TO BE PRINTED/i", line)
then
do;
    message = "FORMAT WAS TOO SMALL FOR THE NUMBER TO BE PRINTED";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/INVALID ARGUMENT/i", line)
then
do;
    message = "INVALID ARGUMENT";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/INVALID NUMERIC DATA/i", line)
then
do;
    message = "INVALID NUMERIC DATA";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/LOST CARD/i", line)
then
do;
    message = "LOST CARD";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/MATHEMATICAL OPERATIONS COULD NOT BE PERFORMED/i", line)
then
do;
    message = "MATHEMATICAL OPERATIONS COULD NOT BE PERFORMED";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/MISSING VALUES WERE GENERATED/i", line)
then
do;
    message = "MISSING VALUES WERE GENERATED";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/MORE THAN ONE DATA SET WITH REPEATS OF BY VALUES/i", line)
then
do;
    message = "MORE THAN ONE DATA SET WITH REPEATS OF BY VALUES";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/NOT FOUND/i", line)
and prxmatch("/Pinnacle Finding/i", line) = 0
then
do;
    message = "NOT FOUND";
    log_line = _n_;  
    output __log_messages;
    goto __end;
end;

if prxmatch("/NOT PREVIOUSLY/i", line)
then
do ;
message = "NOT PREVIOUSLY" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "NOTE: FORMATTED VALUES OF/i" , line )
then
do ;
message = "NOTE: FORMATTED VALUES OF" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "ONE OR MORE LINES WERE TRUNCATED/i" , line )
then
do ;
message = "ONE OR MORE LINES WERE TRUNCATED" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "OUTSIDE THE AXIS RANGE/i" , line )
then
do ;
message = "OUTSIDE THE AXIS RANGE" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "SAS WENT TO A NEW LINE/i" , line )
then
do ;
message = "SAS WENT TO A NEW LINE" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "SEGMENTATION VIOLATION/i" , line )
then
do ;
message = "SEGMENTATION VIOLATION" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "THE MEANING OF AN IDENTIFIER AFTER A QUOTED STRING MAY CHANGE/i" , line )
then
do ;
message = "THE MEANING OF AN IDENTIFIER AFTER A QUOTED STRING MAY CHANGE" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "UERROR/i" , line )
then
do ;
message = "UERROR" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "UNINITIALIZED/i" , line )
then
do ;
message = "UNINITIALIZED" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "UWARNING/i" , line )
then
do ;
message = "UWARNING" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end ;

if prxmatch( "VALUES HAVE BEEN CONVERTED TO/i" , line )
then
do ;
message = "VALUES HAVE BEEN CONVERTED TO" ;
log_line = _n_ ;
output __log_messages ;
goto __end ;
end;

if prxmatch("/ILLEGAL/i", line) then
do;
    message = "ILLEGAL";
    log_line = _n;
    output __log_messages;
    goto __end;
end;

if prxmatch("/SHIFTED BY THE "BEST" FORMAT/i", line) then
do;
    message = 'SHIFTED BY THE "BEST" FORMAT';
    log_line = _n;
    output __log_messages;
    goto __end;
end;

if prxmatch("/NOTE: The variable label/i", line) then
do;
    message = 'NOTE: The variable label';
    log_line = _n;
    output __log_messages;
    goto __end;
end;

if prxmatch("/Pinnacle Finding:/i", line) then
do;
    message = 'PINNACLE FINDING:'
    log_line = _n;
    output __log_messages;
    goto __end;

__end:
if end then output __records;
run;

%if &print_all. = Y
%then
  %do;
    data __messages;
    length message $ 100;
    message = "WARNING";
    output;
    message = "ABNORMALLY TERMINATED";
    output;
    message = "ALREADY EXISTS";
    output;
    message = "ARGUMENT TO FUNCTION"
    output;
    message = "COULD NOT BE WRITTEN";
    output;
    message = "DIVISION BY ZERO DETECTED"
    output;
    message = "ERROR:";
    output;
    message = "ERROR\d-\d+"
    output;
    message = "EXPERIMENTAL IN RELEASE"
    output;
    message = "FORMAT WAS TOO SMALL FOR THE NUMBER TO BE PRINTED"
    output;
    message = "INVALID ARGUMENT"
    output;
    message = "INVALID NUMERIC DATA"
    output;
    message = "LOST CARD"
    output;
  %end;
message = "MATHEMATICAL OPERATIONS COULD NOT BE PERFORMED";
output;
message = "MISSING VALUES WERE GENERATED";
output;
message = "MORE THAN ONE DATA SET WITH REPEATS OF BY VALUES";
output;
message = "NOT FOUND";
output;
message = "NOT PREVIOUSLY";
output;
message = "NOTE: FORMATTED VALUES OF";
output;
message = "ONE OR MORE LINES WERE TRUNCATED";
output;
message = "OUTSIDE THE AXIS RANGE";
output;
message = "SAS SYSTEM STOPPED PROCESSING THIS STEP";
output;
message = "SAS WENT TO A NEW LINE";
output;
message = "SEGMENTATION VIOLATION";
output;
message = "SHIFTED BY THE "BEST" FORMAT";
output;
message = "THE MEANING OF AN IDENTIFIER AFTER A QUOTED STRING MAY CHANGE";
output;
message = "UERROR";
output;
message = "UNINITIALIZED";
output;
message = "UWARNING";
output;
message = "VALUES HAVE BEEN CONVERTED TO";
output;
message = "_ERROR_=1";
output;
message = "ILLEGAL";
output;
message = "NOTE: The variable label"
output;
message = "PINNACLE FINDING:"
output;
run;
%end;
%if %sysfunc(prxmatch(/Y|YES/i, &report.))
%then
%do;
proc sql;
create table __report as
%if &print_all. = Y
%then
select a.message,
      b.log_line,
      b.line,
      case when b.log_line = . then 0
           else 1
      end as found
from __messages               as a
   left join __log_messages as b
      on a.message = b.message
order by a.message,
        b.log_line
; /* end of %if */
%else
select message,
      log_line,
      line,
      1 as found
from __log_messages
order by message,
        log_line
; /* end of %else */
data __report;
set __report where=(found = 1);
run;

proc sort data=__report;
by log_line message;
run;
%end;
%if %sysfunc(prxmatch(/Y|YES/i,&print.)) %then %do;
  data _null_; file print;
  set __report;
  put "Records read from the log: ";
  put "Line = " log_line;
  put line $160. /;
  run;
%end;
%end;
%if %sysfunc(prxmatch(/Y|YES/i,&report.)) %then %do;
  data _null_; file print;
  if nobs = 0 then put "***** No messages found  *****";
  set __report nobs = nobs;
  run;
  %end;
%end;
%if %nrbquote(&print_file.) ne %str() and %sysfunc(prxmatch(/Y|YES/i,&print.)) %then proc printto;
%end;
%__END:
%mend mac_u_log_check;
Appendix 5. The MAC_U_PATH_LOG_CHECK macro.

```sas
%macro mac_u_path_log_check
  (in_ds                = %str(),
   in_ds_keep           = log,
   in_ds_rename         = log = pathfile,
   path                 = %str(),
   path_delim           = #,
   print_messages_all   = N,
   print_messages_gt_0  = N,
   delete               = Y,
   help                 = N)

%if &help. = Y
  %then
    %do;
    %let mprint_orig = %sysfunc( getoption( mprint ));
    options nomprint ;
    skip ;
    skip ;
    %put
    ________________________________________________________________
    _____________________________
    ________________________________________________________________
    %put Purpose of program:     This macro performs a log check on the .log files in the directories provided by the user.
    %put
    _____________________________________________________________________________________________
    %put in_ds                 = Data set contain path and file or pathfile.
    %put %str(                           )Default: %nrstr(%%)str%str(()
    %put path                  = The paths to the log files delimited by the value in PATH_DELIM.
    %put %str(                           )Default: %nrstr(%%)str%str(()
    %put path_delim            = The delimiter of the paths in PATH.
    %put %str(                           )Default: #
    %put print_messages_all    = Whether to print each LINE for the log check for each log file.
    %put %str(                           )Default: N
    %put print_messages_gt_0   = Whether to print each LINE for the log check for each log file that has at least one message.
    %put %str(                           )Default: N
    %put delete                = Whether to delete the data set holding the lists of log files.
    %put %str(                           )Default: Y
    %put
    _____________________________________________________________________________________________
    %put End of help
    %goto __END ;
  %end ;
%if     &in_ds.           = %str()
  and %nrbquote(&path.) = %str()
%then
  %do;
  %put W%str(ARNING: ) IN_DS and PATH cannot both be missing (null) ;
  %goto __END ;
%end ;
%if     &in_ds.           ne %str()
  and %nrbquote(&path.) ne %str()
%then
  %do;
  %put W%str(ARNING: ) IN_DS and PATH both are non-missing.  Please provide a value for only one.
  %goto __END ;
%end ;
%if %nrbquote(&path.) ne %str()
%then %let __ds = dir_read ;
%else %let __ds = &in_ds. ;
/***/
%let byline_orig = %sysfunc( getoption( byline ));
options nobyline ;
%if %nrbquote(&path.) ne %str()
%then %let __ds = dir_read ;
%else %let __ds = &in_ds. ;
*/*/
%let byline_orig = %sysfunc( getoption( byline ));
options nobyline ;
%if %nrbquote(&path.) ne %str()
%then
  %do;
  %mac_u_directory_read
  ( path               = apath,
    path_delim         = apath_delim,
    recursive          = N,
    output_dir         = N,
    output_files       = Y
  );
%end ;
%mac_u_delete
( ds = log_check_all
 log_messages_all
 );
proc sql noprint ;
select pathfile
  into : pf1 -
```

%if &in_ds. ne &str() and %nrbquote(&in_ds_rename.) ne &str()
then %do ;
  %if %nrbquote(&in_ds_keep.) ne &str() %then keep = log  
  rename = ( &in_ds_rename. )
%end ;
where prxmatch("/\./.log$/i", trim( pathfile ))
;
quit ;
%if %nrbquote(spath.) ne &str()
then %do ;
  %mac_u_delete ( ds = &__ds. ) ;
%end ;
%do __i = 1 %to &sqlobs. ;
  %mac_u_log_check ( log_filename         = &&pf&__i.
report               = N
print                = N
print_file           = &str()
delete               = 0
) ;
proc sql
noprint;
select count( * ) into : messages from __log_messages ;
quit ;
proc sql
noprint;
select count( * ) > 0 into : errs from __log_messages
where prxmatch("/ERROR:/", line )
or prxmatch("/ERROR:\d+\-\d+/", line )
or prxmatch("/_ERROR_=1/", line )
or prxmatch("/UERROR/i", line )
;
quit ;
proc sql
noprint;
select count( * ) > 0 into : warns from __log_messages
where prxmatch("/WARNING/", line )
and prxmatch("/UWARNING/", line ) = 0
and prxmatch("/WARNING: Engine XPORT does not support SORTEDBY operations. SORTEDBY information cannot be copied./", line ) = 0
and prxmatch("/WARNING: Some character data was lost during transcoding in column/", line ) = 0
and prxmatch("/WARNING: The quoted string currently being processed has become more than 262 characters long./", line ) = 0
;
quit ;
proc sql
noprint;
select count( * ) > 0 into : stops from __log_messages
where prxmatch("/ABNORMALLY TERMINATED/i", line )
or prxmatch("/ENDSAS/i", line )
or prxmatch("/SAS SYSTEM STOPPED PROCESSING THIS STEP/i", line )
or prxmatch("/SEGMENTATION VIOLATION/i", line )
;
quit ;
/*******

data __log_check ;
merge info
__records
__log_messages ;
messages = $messages. ;
log_has_message = messages > 0 ;
log_has_warning = $warns. ;
log_has_error   = $errs.  ;
log_has_stops   = $stops. ;
run ;
%mac_u_delete ( ds = __records 

%if &messages. = 0
%then
%do;
  data __log_messages;
  if 0 then set __log_messages;
  message = "No messages found";
  line = "No messages found";
  output;
  stop;
run;
%end;

proc sql
  undo_policy = none;
create table __log_messages as
  select a.path, a.file, b.*
  from finfo as a, __log_messages as b
  order by a.path, a.file, b.message, b.log_line;
quit;

%mac_u_delete (ds = finfo)
/***/
%if %sysfunc( exist( log_messages_all ) )
%then
  %do;
    proc append
      base = log_messages_all
data = __log_messages;
run;
  %end;
  %else
    proc datasets
      library = WORK
      nolist;
    delete __log_messages;
    quit;
  %end;
%end; /* CYCLED THROUGH __i */

proc sort
data = log_check_all;
by path, file;
/**/
%if %sysfunc( exist( log_check_all ) )
%then
  %do;
    proc append
      base = log_check_all
data = __log_check;
run;
  %end;
  %else
    proc datasets
      library = WORK
      nolist;
    delete __log_check;
    quit;
  %end;
%end; /* CYCLED THROUGH __i */

proc sort
data = log_check_all;
by path, file;
run;

proc sort
  data = log_messages_all;
  by path
    file
    log_line
  ;
run;

/**********/

footnote;

title "path = #byval1";

proc print
  data = log_check_all
  noobs
  summlabel = "Total in path:"
  grandtotal_label = "Grand total in all paths:"
  n = "Logs in path:"
  "Total Logs in all paths:"
  ;
  by path;
  sum messages
    log_has_message
    log_has_warning
    log_has_error
    log_has_stops
  ;
run;

title "";

%if &print_messages_all. = Y
  %then
    %do;
    title1 "path = #byval1";
    title2 "file = #byval2";
    proc print
      data = log_messages_all
      noobs
      ;
    by path
      file
      ;
    var log_line
    line
    ;
    format line $240.;
run;
    title "";
    %end;
%end;

%if &print_messages_gt_0. = Y
  and &print_messages_all. = N
  %then
    %do;
    proc sql
      noprint;
    select distinct quote( strip( catx( "", path, file )))
      into : list separated by "", "" from log_check_all
    where messages > 0
    ;
    quit;
    title1 "path = #byval1";
    title2 "file = #byval2";
    proc print
      data = log_messages_all
      ( where = ( catx( "", path, file ) in
      ( &list. ) )
      )
      noobs
      ;
    by path
      file
      ;
    var log_line
    line
    %end;

34
378 \};
379   format line $240. ;
380   run ;
381   title "" ;
382   %end ;
383   options $byline_orig. ;
384   %if %sysfunc( upcase( &delete. )) = Y
385     %then
386     %do ;
387     %mac_u_delete
388     ( ds = $_ds. )
389     %end ;
390   %END:
391   %end  mac_u_path_log_check ;
392
393
Appendix 6. The MAC_R_COMPARE macro.

```
%macro mac_r_compare_report
  (path = %str(),
   file = %str('*.*'),
   filename = %str(),
   out = %str('_compare'),
   print = Y,
   print_vars = file,
   data = comp,
   data_last_modified = comp_last_modified,
   datetime_last_modified = datetime_last_modified,
   data_vars = vars_in_common,
   data_obs = obs_in_common,
   obs_issue = obs_issue,
   obs_w_unequal = obs_w_unequal,
   obs_all_equal = obs_all_equal,
   print_where = %str(),
   print_vars = file,
   print = Y,
   out = __compare,
   filename = %str(),
   file = *.lst,
   path = %str()) ;

%if %eq(&help, Y)
%then
%if &print. = N
%then
%if &mv_fail = compare_fail
%then
%if &print_sum = obs_w_unequal
%then
%if &print_where = %str()
%then
%if &print_vars = file
%then
%if &print = Y
%then
%if &out = __compare
%then
%if &filename = %str()
%then
%if &file = *.lst
%then
%if &path = %str()
%then

The MAC_R_COMPARE macro.

<table>
<thead>
<tr>
<th>Macro Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose of program</td>
<td>This report macro parses .lst files extracting pertinent COMPARE data.</td>
</tr>
<tr>
<td>The following shows</td>
<td>what data are extracted with their SAS variable names in parentheses:</td>
</tr>
<tr>
<td>The COMPARE Procedure</td>
<td></td>
</tr>
<tr>
<td>Comparison of</td>
<td>VALDS.T14_1_9_62_1 with WORK.V_T14_1_9_62_1</td>
</tr>
<tr>
<td>(Method=EXACT)</td>
<td></td>
</tr>
<tr>
<td>Variables Summary</td>
<td></td>
</tr>
<tr>
<td>Number of Variables in Common: 14. (VARS_IN_COMMON)</td>
<td></td>
</tr>
<tr>
<td>Number of ID Variables: 6.</td>
<td></td>
</tr>
<tr>
<td>Observation Summary</td>
<td></td>
</tr>
<tr>
<td>Dataset Created</td>
<td>Modified NVar NObs</td>
</tr>
<tr>
<td>First Obs PARAM=Alanine Aminotransferase (U/L) ORD=1 TRTN=1 AVISITN=2 DAY=Baseline</td>
<td></td>
</tr>
<tr>
<td>Last Obs 72 PARAM=Urea Nitrogen (mmol/L) ORD=4 TRTN=4 AVISITN=4 DAY=Day 15</td>
<td></td>
</tr>
<tr>
<td>Number of Observations in Common: 72. (OBS_IN_COMMON)</td>
<td></td>
</tr>
<tr>
<td>Total Number of Observations Read from VALDS.T14=1-9: 72. (OBS_DATA)</td>
<td></td>
</tr>
<tr>
<td>Total Number of Observations Read from WORK.V_T14=1-9: 72. (OBS_COMP)</td>
<td></td>
</tr>
<tr>
<td>Number of Observations with Some Compared Variables Unequal: 0. (OBS_W_UNEQUAL)</td>
<td></td>
</tr>
<tr>
<td>Number of Observations with All Compared Variables Equal: 72. (OBS_ALL_EQUAL)</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: No unequal values were found. All values compared are exactly equal.
```
%if %nxquote(&path.) ne "" %then %let filename = "&path." ;
%end ;
%else &out. ;
%end ;
__rc8 = prxparse( "/Number of Observations with All Compared Variables Equal: (" );
__rc7 = prxparse( "/Number of Observations with Some Compared Variables Unequal: (" );
__rc6 = prxparse( "/Number of Variables in Common: (" );
__rc5 = prxparse( "Comparison of (" );
__rc4 = prxparse( "") or  obs_w_unequal > 0
__rc3 = prxparse( ") or  data_obs      ne obs_in_common
__rc2 = prxparse( ") or  comp_vars     ne vars_in_common
__rc1 = prxparse( ")or  comp_obs      ne obs_in_common

/************/
data out. ; ( drop = _1 ) ;
if _n_ = 1 then
  _rc1 = prxparse("/Comparison of ([_A-Z][_A-Z]+[0,1]([_A-Z]+[0,1]) with ([_A-Z][_A-Z]+[0,1]([_A-Z]+[0,1]))")
  _rc2 = prxparse("/[Number of Variables in Common: (\d+)\/^/\"]")
  _rc3 = prxparse("/Number of Observations with Some Compared Variables Unequal: (\d+)\/^/\"]")
  _rc4 = prxparse("/Number of Observations with All Compared Variables Equal: (\d+)\/^/\"]")
  _rc5 = prxparse("/Number of Observations with Some Compared Variables Unequal: (\d+)\/^/\"]")
  _rc6 = prxparse("/Number of Variables in Common: (\d+)\/^/\"]")
  _rc7 = prxparse("/Number of Observations with Some Compared Variables Unequal: (\d+)\/^/\"]")
  _rc8 = prxparse("/Number of Observations with All Compared Variables Equal: (\d+)\/^/\"]")
end ;

length file $ 200
data comp
  _data = _data
  _comp = _comp
  _comp_last_modified $ 41
data_vars
data_obs
data_last_modified
data_vars
comp_last_modified
comp_vars
comp_obs
comp_vars
comp_obs
```plaintext
retain __flag1 " "
__flag2 "1"
__flag3 " 
__rc1
__rc2
__rc3
__rc4
__rc5
__rc6
__rc7
__rc8 .
data
comp
data
__data
comp
__data
__comp
data_last_modified
data_vars
data
obs
__comp
comp_last_modified
comp_vars
comp
vars_in_common
obs_in_common
obs_data
obs_comp
obs_w_unequal
obs_all_equal
__fn1
infile &filename.
length   = len
filename = __fn
end      = __end
;
if __fn1 = " " then __fn1 = __fn
else if __fn1 ne __fn
then
do;
if __flag3 = " 
then
do;
    pathfile = __fn1
    file = scan( pathfile , -1 , "\n" )
    output
    end
else __flag3 = " 
__fn1 = __fn
end
input __line $varying256.
len ;
if prxmatch( "/The COMPARE Procedure/" , __line ) then __flag1 = "1" ;
if __flag1 = "1"
then
do;
    if prxmatch( __rc1 , __line )
then
do;
        data = prxposn( __rc1 , 1 , __line )
        comp = prxposn( __rc1 , 2 , __line )
        __rc5 = prxparse( cat( "/Total Number of Observations Read from "
            , strip( data )
        )
    );
    __rc6 = prxparse( cat( "/Total Number of Observations Read from "
            , strip( comp )
        )
    );
end
input __line $varying256.
len ;
if prxmatch( "/The COMPARE Procedure/" , __line ) then __flag1 = "1" ;
if __flag1 = "1"
then
do;
    if prxmatch( __rc1 , __line )
then
do;
        data = prxposn( __rc1 , 1 , __line )
        comp = prxposn( __rc1 , 2 , __line )
        __rc5 = prxparse( cat( "/Total Number of Observations Read from "
            , strip( data )
        )
    );
    __rc6 = prxparse( cat( "/Total Number of Observations Read from "
            , strip( comp )
        )
    );
end
if data ne __data then put "ERR" "ROR: data mismatch: " data= __data= ;
```
input _line $Varying256. len;
if prxmatch( _rc2 , _line )
then
do;
__comp = prxposn( _rc2 , 1 , _line );
__comp_last_modified = input( prxposn( _rc2 , 2 , _line ) , datetime );
comp_vars = input( prxposn( _rc2 , 3 , _line ) , 8. ) ;
comp.obs = input( prxposn( _rc2 , 4 , _line ) , 8. ) ;
if comp ne __comp then put "ER" "ROR: comp mismatch: " comp= __comp= ;
end;
if prxmatch( _rc3 , _line ) then vars_in_common = input( prxposn( _rc3 , 1 , _line ) , 8. ) ;
if prxmatch( _rc4 , _line ) then obs_in_common = input( prxposn( _rc4 , 1 , _line ) , 8. ) ;
if __rc5 ne .
and prxmatch( _rc5 , _line )
then obs_data = input( prxposn( _rc5 , 1 , _line ) , 8. ) ;
if __rc6 ne .
and prxmatch( _rc6 , _line )
then obs_comp = input( prxposn( _rc6 , 1 , _line ) , 8. ) ;
if prxmatch( _rc7 , _line ) then obs_w_onequal = input( prxposn( _rc7 , 1 , _line ) , 8. ) ;
if prxmatch( _rc8 , _line )
or prxmatch("/Comparisons of data values not performed./", _line )
then
do;
if prxmatch( _rc8 , _line ) then obs_all_equal = input( prxposn( _rc8 , 1 , _line ) , 8. ) ;
pathfile = fn :
file = scan( pathfile , -1 , "\" );
if data_last_modified ne .
and comp_last_modified ne .
and comp_last_modified < data_last_modified
then datetime_issue = "Y" ;
if data_vars ne .
and vars_in_common ne .
and data_vars ne vars_in_common
then vars_issue = "Y" ;
if data_obs ne .
and obs_in_common ne .
and data_obs ne obs_in_common
then obs_issue = "Y" ;
output ;
call missing(file , data , comp ,
data_last_modified , data_vars ,
data_obs , _comp ,
comp_last_modified , comp_vars ,
comp.obs , vars_in_common , obs_in_common ,
obs_data , obs.comp , obs.w_onequal , obs.all_equal ,
datetime_issue , __flag2
g) ;
__flag3 = "1" ;
end; /* END OF __flag1 = "1" */
if __end = 1
and __flag3 = "1"
then
do;
pathfile = fn :
file = scan( pathfile , -1 , "\" ) ;
output ;
end;
format data_last_modified comp_last_modified datetime20.
run;
proc sort
data = &out. ;
by file
data ;
run;
%if &print. = Y
%then
do;
%if %nrbquote(&path.) ne %str() %then title "&path." %str(;) ;
%else title " " %str(;) ;
proc print
   data = &out. ;
   %if %nrbquote(&print_where.) ne %str() %then ( where = ( &print_where. )) ;
   %if &print_vars. ne %str() %then var &print_vars. %str(;) ;
   %if &print_sum. ne %str() %then sum &print_sum. %str(;) ;
run ;
%if %nrbquote(&path.) ne %str() %then title1 "&path." %str(;) ;
%else title1 " " %str(;) ;
title2 "COMPARE results with matching variable and observations numbers and no unequal values" ;
proc sql ;
   select count( * )
   from &out.
   where data ne ""
   and data_vars = vars_in_common
   and data_obs = obs_in_common
   and obs_w_unequal = 0;
quit ;
%end ;
%let &mv_fail. = 0 ;
data _null_ ;
set __compare ;
if data ne ""
   and ( data_vars ne vars_in_common
   or comp_vars ne vars_in_common
   or data_obs ne obs_in_common
   or comp_obs ne obs_in_common
   or obs_w_unequal > 0 )
then do ;
call symput( "mv_fail." , "1" ) ;
   stop ;
end ;
run ;
%END:
%end mac_r_compare_report ;
Appendix 7. The MAC_U_DELETE macro.

`%macro mac_u_delete
  (library = WORK, ds = , help = N);
%if &help. = Y
%then
  %do;
  %let mprint_orig = %sysfunc(getoption(mprint));
  options nomprint;
  skip;
  skip;
  %put _____________________
  _______________________________________________________
  ________________________________;
  %put Purpose of program: This utility macro uses the DATASETS procedure to delete (a list of) SAS data sets, if they exist.
  %put _________________       _______________________________________________________
  %put library               = The libref of the data(s) to delete
  %put ds                    = The name or the list of (one- or two-level) names of data set(s) to delete.
  %put End of help
  options &mprint_orig.
%end;
%local i d library ds;
%if %index( &ds. , . ) = 0
%then
  %do;
  %let i = 1;
  %let d = %scan( &ds. , &i. , %str( ))
  %do %while ( &d. ne )
    %if %sysfunc( exist( &library..&d. )) %then delete &d. %str(;) 
    %let i = %eval( &i. + 1 )
    %let d = %scan( &ds. , &i. , %str( ))
  %end;
  quit;
%end;
%else
  %do;
  %let i = 1;
  %let d = %scan( &ds. , &i. , %str( ))
  %do %while ( &d. ne )
    %if %sysfunc( index( &d. , . )) = 0 %then %let d = &library..&d. 
    %if %sysfunc( exist( &d. )) %then
      %do;
      proc datasets library = %if %index( &d. , . ) %then %scan( &d. , 1 , . ) ;
      %else &library. ;
      delete %if %index( &d. , . ) = 0 %then %let d = &library..&d. ;
    %end;
    %let i = %eval( &i. + 1 )
    %let d = %scan( &ds. , &i. , %str( ))
  %end;
  quit;
%end;
%end;
%__END: %mend  mac_u_delete ;