PhUSE 2016

Paper CS08

CompareDirectory – Macro bits and pieces in action

Lawrence Heaton-Wright, QuintilesIMS, Reading, UK

INTRODUCTION

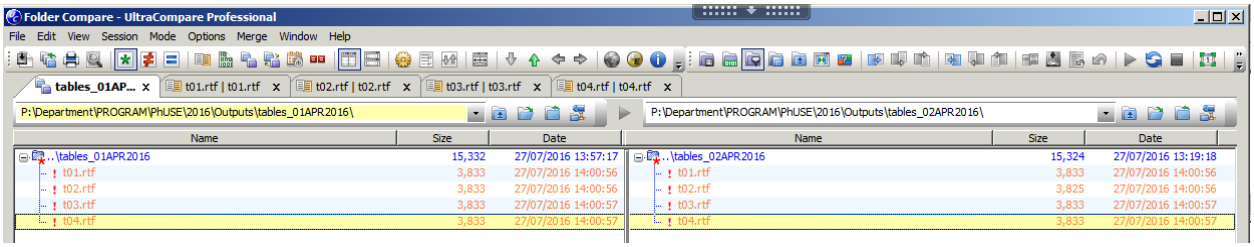
CompareDirectory is a macro written to check all common files across 2 directories to identify which, if any, files are different. There is an option to exclude a line from this comparison (i.e. the date/time of production). This macro was designed to simplify the process of checking outputs between deliveries so that only the files that are different can be identified. The macro does not produce any detailed file comparison output but instead shows a report identifying the files that are different as well as identifying files that do not exist in both directories.

The macro was written because the task of checking re-runs of outputs can be quite a tedious manual process. There is quite a lot of file difference checking software available (such as UltraCompare, ExamDiff, etc.) and these generally have folder comparison functionality as well.

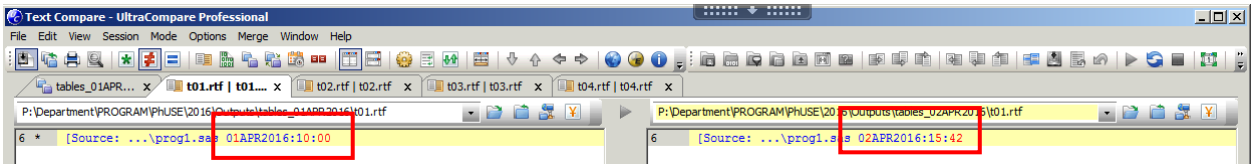
The problem with SAS outputs for clinical trials is that we usually have the date/time of production included somewhere in the output (please note this is not a problem except from a point of view of comparing outputs). If the entire study is re-run because of, for instance, data changes then checking the 2 sets of outputs using file comparison software will show that all outputs are different because of the date/time of production. However on more detailed inspection we can see that the only thing in many of these outputs that is different is the date/time.

THE PROBLEM

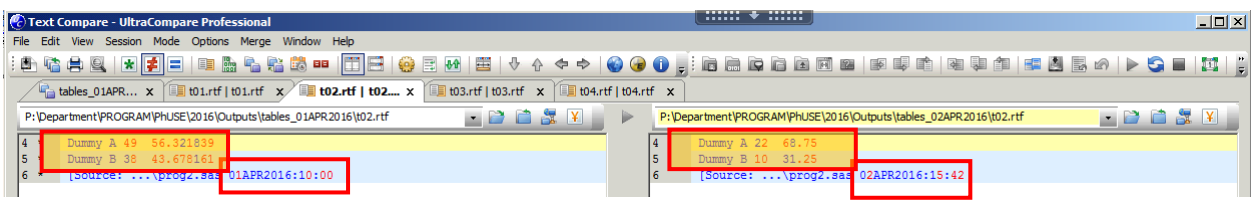
Identifying what outputs are actually different and which ones were just re-run. The example given below is only 4 outputs, yet we need to check each one of them to ascertain which one is truly different. This involves utilising the differencing software for a folder compare, then opening every one of the files that shows some difference.



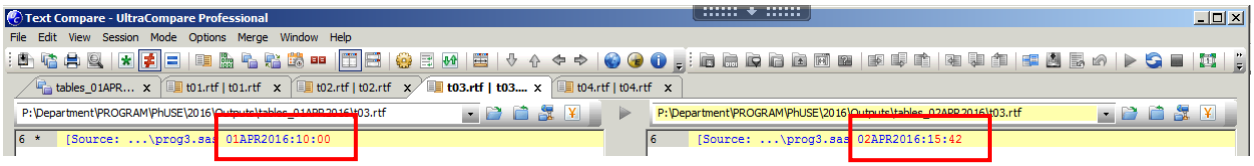
As can be seen from the first compare that has been performed the only difference is the date of production.



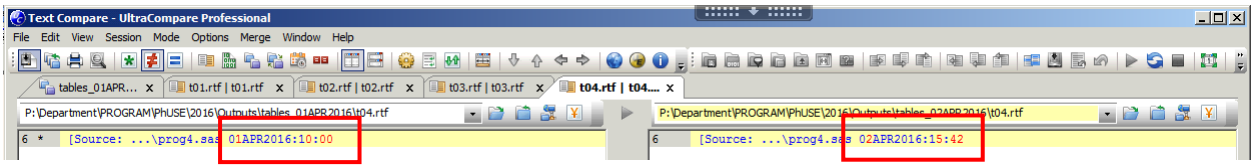
The second output we compare actually has some differences in the numbers produced as well as the date of production. This is one of the outputs we are looking for. Now we've identified this we can go through the validation processes as well as checking that this difference is what is expected.



The third output compared also only has the date of execution being different.



The fourth file compared only has a difference in the date of production.



This only took about 10 minutes as there are only 4 outputs (including checking the validation, etc.). If, however, there were 200 outputs then it could take about 2 hours (loosely based on 30 seconds per compare plus validation checking). It could take as much as 3 or 4 hours if they are big outputs. Clearly this is not an optimum use of time as well as being a tedious process which could lead to mistakes being made.

THE SOLUTION

What we need is a method of directing us to the outputs with actual differences in them to enable the project team to check that the differences in outputs are those expected or have been validated correctly rather than having to repetitively open and close the individual file comparison screens. After all this is what a computer program does best!

Being a SAS programmer my mind turned to using SAS. After all, the output files (even RTF and DOC files) are just text files. These files could be read into SAS datasets and these datasets compared to see if anything had changed excluding a string or phrase from consideration.

By coincidence I had been investigating SAS external file functions (PhUSE 2013 paper CS02). The outputs produced were external files hence a good chance to use these functions.

This task was suited to macro processing as I would need to carry out the same task on 2 different locations. But there was a need for speed as I only had a couple of hours to put the draft program in place so I couldn't spend lots of time error-trapping and programming as defensively as I normally would as it required a swiftly-produced bespoke program.

IDENTIFY FILES IN COMMON

The first thing that was required is to in 2 outputs (the “same” output, one from each file location). These were read into 2 SAS datasets. The _N_ variable was defined as OBSNO (observation number) which is then used as the BY variable in the DATA step MERGE. These datasets were then merged together and the IN= dataset option was used to identify whether any records did not match. If they didn't then something was different between the outputs. As expected this difference was the date/time stamp from the initial 2 outputs I compared as I was just testing the concept.

The next step was to ascertain a way of uniquely identifying this record which would work across all outputs and would not exclude actual data. I couldn't just look for a date/time string using a Perl comparison string as date/times appear in outputs as data and we wanted to see if any data had changed. We needed something that would identify the record uniquely.

Based on the knowledge of the project I knew that if the footnote started with a certain defined string this would identify the date/time stamp. By identifying this record using the INDEX function I could delete this record(s) from each "output" dataset and then perform the data merge as described above. Removing this string showed that these initial test outputs were the same once the date/time stamp of production had been removed from the comparison process.

Now I had to expand the program so that I could read in the folder information and automate the reading in of outputs and the comparison process excluding the date/time stamp. This is where the external file functions come in.

```
%LET filrf=MYDIR;
%LET rc=%SYSFUNC(FILENAME(filrf,"&&Dir&i.."));
%LET did=%SYSFUNC(DOPEN(&filrf));
%LET memcount=%SYSFUNC(DNUM(&did));
%LET rc=%SYSFUNC(DCLOSE(&did));
```

Use the FILENAME function to assign a file reference to MYDIR (when FILENAME is used within a %SYSFUNC then the file reference name is the name of a macro variable without the &. In this case FILRF). Use the DOPEN command to open the specified directory. The DNUM function then returns the number of objects within this directory. This is stored in the macro variable MEMCOUNT. Once this information is retrieved then the DCLOSE function is used to close the directory.

This information is then used to iterate through every single object within the target directory, read the filename information into a DATA step, check whether the file has the required file extension and if it does send this record (detailing the directory and filename) to the “output” dataset.

```
DATA _SASFiles_&i. (DROP=_:);
  ATTRIB
    __FileExt   LENGTH=$15.
    SASFile     LABEL="SAS File" LENGTH=$100.
    Dir&i.      LABEL="Directory &i." LENGTH=$200.
  ;
*** Open directory ***;
  __rc=filename("mydir","&&Dir&i..");
  __did=dopen("mydir");
  DO __i=1 TO &memcount;
    SASFile=DREAD(__did,__i);
    Dir&i.="&&Dir&i..";
    __FileExt=UPCASE(SCAN(SASFile,-1,'.'));
    IF __FileExt EQ "&FileType." THEN OUTPUT;
  END; *** __I ***;
RUN;
```

This code is contained within a macro DO loop to scan through both directories, hence the double ampersands and presence of &I within the DATA step. The DREAD function reads the name of the directory member (filename and file extension) into SASFILE. The file extension is extracted from this, compared to the required file extension type and only kept if these match.

The FILETYPE macro variable is already set to uppercase at the start of the macro using the %UPCASE macro function (%LET FileType=%UPCASE(&FileType);). I use this construct in most of the macros I write to avoid case-matching problems. This is especially useful if you have user-defined macro variables in the macro call and you need to use these macro variables in comparisons or selection clauses.

```
DATA
  SAS_Files (keep=SASFile Dir:)
  _FilesNotMatching (keep=SASFile inDir)
  ;
  MERGE
    _SASFiles_1 (IN=in_dir1)
    _SASFiles_2 (IN=in_dir2)
  ;
  BY SASFile;
  IF in_dir1 AND in_dir2 THEN DO;
    OUTPUT SAS_Files;
  END;
  ELSE DO;
*** Used for report detailing files in one directory and not the other ***;
    if in_dir1 then inDir='File only in Directory 1';
    if in_dir2 then inDir='File only in Directory 2';
    output _FilesNotMatching;
  END;
RUN;

DATA SAS_Files;
  SET SAS_Files END=eof;
  ATTRIB
    _file_id  LABEL="File ID Value"
  ;
  BY SASFile;
  _file_id=_N_;
  IF eof THEN DO;
    CALL SYMPUTX ('N_File',_file_id,'L');
  END;
RUN;
```

Once the list of files in each folder was obtained these 2 datasets were merged together and those filenames that are in common were kept. An ID variable was created using the _N_ value resultant from the SET statement and the total number of files in common was sent to a macro variable (N_FILE) to use as the upper limit of a macro DO loop. To ensure that this macro variable is not created for every record from this dataset the END= SET statement option is used to create a flag indicating that the last record from the input dataset has been reached. When this flag is valid then this indicates this is the last record and the CALL SYMPUTX is used to create the N_FILE macro variable.

```
%LET dsid=%SYSFUNC(OPEN(work._FilesNotMatching));
%LET FNM_nObs=%SYSFUNC(ATTRN(&dsid.,NOBS));
%LET rc=%SYSFUNC(CLOSE(&dsid.));
```

In the subsequent versions of the macro a macro variable is defined which contains the number of files which are not in common. This can then be used to determine whether this report should be printed or not. The OPEN, ATTRN and CLOSE functions were used to determine the number of files in common. These functions are an extremely neat and efficient method for selecting this kind of dataset metadata. The files not in common were originally simply removed from the dataset (as we had no interest in these files). In the later version these files were sent to another dataset and the information presented to show files in one folder but not in the other. In theory this should produce no data but if it does then the programming lead can utilise this information to clean up the project folders if required.

DETERMINING THE DIFFERENCES

The dataset comprising the files in common (SAS_FILES) contains the 2 directory locations and the filename which is to be compared. The total number of files has been determined earlier and sent to a macro variable (N_FILE). This can then be used to iterate through every file to be compared. The control dataset (SAS_FILES) is then used to create 2 local macro variables (FILE1 and FILE2) for each file. These are then to be used in a FILENAME statement which is then used as the input to a dataset.

```
%LET StatusDSet=;
%IF &N_File. GT 0 %THEN %DO FileNum=1 %TO &N_File.;
  DATA _NULL_;
    SET SAS_Files (WHERE=(_file_id EQ &FileNum.));
    %DO i=1 %TO 2;
      CALL SYMPUTX ("File&i.",CATX("&dlm.",dir&i.,SASFile),'L');
    %END;
  RUN;

  %DO i=1 %TO 2;
    FILENAME SASFile "&&File&i..";
    DATA _File_&i.;
      INFILE SASFile LRECL=200 TRUNCOVER;
      INPUT @1 line&i. $CHAR200.;
    RUN;
```

Following the reading in of the file to dataset (shown above) we now need to remove the exclusion text record from this dataset. In the original macro this was a defined text string (it was a stand-alone program originally) but in the later versions this was user-defined. If there is text to exclude then use an INDEX function to identify the record and delete this record from the dataset.

```
    %IF &FileType. in (LST TBL RTF) and %bquote(&TextExcl.) ne %THEN %DO;
      DATA _File_&i.;
        SET _File_&i.;
        IF index(UPCASE(line&i.),"%upcase(&TextExcl.)") gt 0 THEN DELETE;
      RUN;
    %END;
```

For RTF files remove the line which contains the file metadata which is based on the line with Creation date/time in it.

```
    %IF &FileType. in (RTF) %THEN %DO;
      DATA _File_&i.;
        SET _File_&i.;
        IF index(UPCASE(line&i.),"CREATIM") gt 0 THEN DELETE;
      RUN;
    %END;
```

The next task is to define a merging variable (OBSNO). The theory is that if the files are identical (after removing the exclusion text) then the observation number will contain the same information for each file. If not this indicates that there is a difference in the files.

```
    DATA _File_&i.;
      SET _File_&i.;
      ATTRIB
        ObsNo   LABEL="SAS File Line Number"
      ;
      ObsNo=_n_;
    RUN;

    filename SASFile;
  %END; *** I for selecting Files ***;
```

Following this stage the 2 datasets containing the read in output are then merged together. Using a simple variable comparison any differences in the outputs are detected and a single observation is sent to a findings dataset. A dataset is created for each pair of files that are being compared. The COMPBL function is used to ignore differences in white space in the outputs.

```
  DATA _CheckDiff;
    MERGE
      _File_1
      _File_2
    ;
    BY ObsNo;
    _file_id=&FileNum.;
    IF COMPBL(line1) NE COMPBL(line2) THEN _difference=1;
  RUN;

  DATA _Status_&FileNum. (KEEP=_file_id AnyDiff);
    SET _CheckDiff END=eof;
    ATTRIB
      AnyDiff   LABEL="Any Differences" LENGTH=$1.
    ;
    RETAIN AnyDiff 'N';
    IF _difference EQ 1 THEN AnyDiff='Y';
    IF eof THEN OUTPUT;
  RUN;
  %LET StatusDSet=&StatusDSet. _Status_&FileNum.;
%END; %*** FileNum ***;
```

The STATUSDSET macro variable is initialised prior to iterating through the pairs of files. This will then have a list of dataset names which is then used to create the overall status dataset of the findings of the pairs of files using a SET statement. This is a construct I use quite regularly in place of a PROC APPEND as this way I find that the dataset created will not need to be removed from the WORK library if the macro is called multiple times within the same program whereas PROC APPEND will continually add to an existing dataset unless the WORK library is cleaned up.

```
DATA Status;
  SET &StatusDSet.;
  BY _file_id;
RUN;
```

REPORT THE DIFFERENCES

Once this status dataset has been created the next task is to check whether there are any differences (a macro variable is defined). If there are differences then a report is generated showing the files where differences occur alongside information showing how many files have differences, how many files were compared, where these files are located and what the text excluded is. If there are any files not in common this information is also reported.

The reporting section shows as titles the 2 directories being compared, the text string that has been excluded, the file extension being compared and the number of files in common and the number of files with differences. This then shows the files that contain actual differences. These files can then be looked at in more detail using the file comparison software in addition to standard validation.

If there are files not in common in the 2 directories (the FILESNOTMATCHING dataset indicated by the FNM_NOBS macro variable having a value greater than zero) then this information is also reported.

The reports are simple PROC PRINTs of files with and without differences as well as a printout of files not in common. An enhancement from the original program was adding a macro variable to identify whether the output should be sent to an RTF file (using ODS RTF) or the output window. The RTF filename has the date of macro execution added (in YYYY-MM-DD format) as an aid to identify when the file comparison was executed.

FUTURE ENHANCEMENTS

Although the macro is widely used there are a couple of enhancements that could be made that would enhance the functionality whilst maintaining the simplicity of the macro call.

- A clean-up macro could be added to clear out WORK datasets and global macro variables that are produced during the execution of the macro. This would mean that instead of using the STATUSDSET macro variable construct a PROC APPEND process could be used which would be slightly less code.
- Add an option to send the reports to an Excel workbook in place of the RTF file/output window.
- Add an option to ignore case for text differences
- Add an option to ignore white space for text differences
    - This is the default behaviour of the macro but could be switched off

CONCLUSION

Developing this macro was an interesting challenge. Although the initial program was developed on a Windows system in-house one of the requirements was for it to work on a customer-based Unix server so any operating system based commands which could be used to ascertain the list of files in a folder were instantly ruled out as the program needed to work on multiple operating systems. I had used external file functions occasionally and remembered from the SAS online help that there were plenty of functions to identify and read in the information about files in folders. So these were at the forefront of my mind when I received the request to create the original program.

The original program probably took about 5 hours to put together including testing and de-bugging. Based on an estimated time of 30 seconds per output to check the comparison software the 5 hours of total development time would be repaid once more than 600 outputs have been compared. As the program works across both Windows and Unix systems this meant we could reuse the program across different projects as and when required and the development cost has been recouped many times over.

CONTACT INFORMATION

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

Lawrence Heaton-Wright
QuintilesIMS
500 Brook Drive,
Green Park,
Reading,
Berkshire.
RG2 6UU
United Kingdom

Work Phone: + 44 118 4508320
Email: lawrence.heaton-wright@quintilesims.com
Web: www.quintilesims.com