

Automated File Comparison and Validation Across Programming Platforms

Padma Kishan Korsapati
Darrell Edgley



March 2026

Purpose of the Paper

Periodic upgrade of IDE's

A method for promote but also consistent verification

Advantages

Validate the base method, repeat as needed. Ensure a quality end-game verification for low net cost

Outcome

A report that can be attested to quickly by the project team, ensuring that the target system files are a reasonable evolution from the source system

Our typical folder tree



Data and Output Folders

Folders such as './adam/', './sdtn/', './analysis', and './tlf's' store datasets, analytical outputs, and key deliverables for the project.



Code and Script Repositories

'./macros/' and './programs' organize reusable code and scripts, supporting efficient and consistent project programming.



Workflow Support Directories

'./logs/' and './tools/' hold logs and utility programs, aiding process tracking and workflow automation.

Our environment is LINUX based, examples within this paper presentation are therefore LINUX, however the principles can be applied to other operating systems/environments

Folder configuration

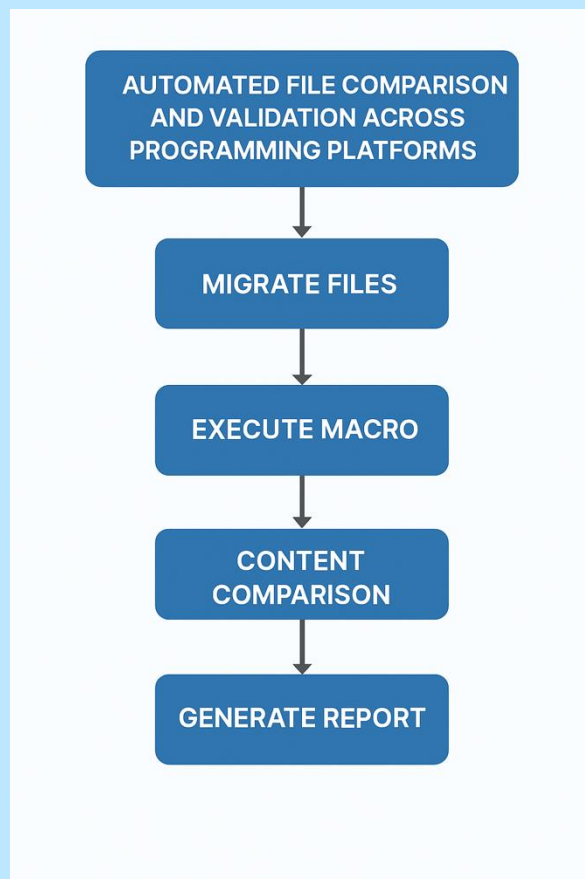
In simple terms within our organization, we have main project folders, then a standard set of sub-folders below that folder

In our recent IDE upgrade, the names of the folders in the source (old) and target (new) system are not the same but there is a correlation.

Our migration tools copy relevant source files to the appropriate target location. In some circumstances the filenames are updated, in some other cases the file type is converted (for example text based parameter files to JSON parameter files)

Technical solution

The %check_for_migration SAS macro



Two use cases:

- 1) Pre-migration. Check the source files for compatibility to standards, highlight discrepancies or items that are “not typical” so that the programming lead can consider ... “would pre-migration mitigation be prudent”
- 2) Post migration. The difference in the macro call is the path for the target environment is included and the utility then checks source against target and reports what it finds.

When executed in scenario 2, the migration has already been performed, we **EXECUTE** the macro, it performs a “**CONTENT COMPARISON**” between source (old) system and the target (new) system and “**GENERATES a REPORT**” for attestation by our study programming lead.

Technical solution: Macro parameters

The %check_for_migration has the following base parameters:

Parameter	Description
BASEPROJ	The base project folder of our historical reporting system
TARGETPROJ	The base project folder of the target or new system. This parameter is optional and its use or not allows for the two use cases: - Not supplied, the code does a pre-migration check of the BASEPROJ area and creates a report for any deltas from standards. This allows at the programming leads discretion for pre-migration mitigation as necessary - When it is supplied the files from BASEPROJ and compared to TARGETPROJ, acknowledging that the files may be in sub-folders in both environments. In our specific situation the sub-folder names are not consistently named, but in this paper the examples will assume that the base and target sub-folders do have the same names

Technical solution: What is checked?

The %check_for_migration has the following base parameters:

File types	Comments
SDTM and ADaM datasets	We perform SAS data set compare using THREE executions of PROC COMPARE. More details on next slide
QC and Production code	Program files, be they SAS (the majority) or others including R. These are evaluated for differences using the LINUX diff utility
Parameter files	Our parameter files in our historical system were all text based, in our new system they are JSON based. We can not directly evaluate these for difference, we therefore read both the source and target files into SAS, align on variable names and then PROC COMPARE the parameter name and parameter values

The misnomer of PROC COMPARE

It is beyond the scope of this paper to delve too deep in to this aspect, but the take home message is that the PROC COMPARE statement of “All values compared are exactly equal” does NOT mean that the TWO data sets are in fact IDENTICAL. To prove two data sets are in fact identical we run PROC COMPARE three times on different aspects of the data.

Query to data dictionary	PROC compares
<pre>%let baseds=old.ae; %let newds=new.ae; %let source_libref=%scan(&baseds,1,.) ; %let source_dsname=%scan(&baseds,2,.) ; %let target_libref=%scan(&newds,1,.) ; %let target_dsname=%scan(&newds,2,.) ; proc sql noprint ; create table work._source_vars as select name, type, length, label from dictionary.columns where libname = "&source_libref" and memname = "&source_dsname" order by name ; create table work._source_dsstruct as select memlabel, nlobs, nvar from dictionary.tables where libname = "&source_libref" and memname = "&source_dsname" order by name ; %* repeat for TARGET data set</pre>	<pre>%* Compare 1 ; proc compare base=&baseds (encoding=asciiany) compare=&newds (encoding=asciiany) out = work._compare_&basedsnm outnoequal ; run ; %* Compare 2 ; proc compare base = work._source_vars compare = work._target_vars out = work._compare_structure outnoequal noprint ; run ; %* Compare 3 ; proc compare base = work._source_dsstruct compare = work._target_dsstruct out = work._compare_dsstruct outnoequal noprint ; run ;</pre>

The misnomer of PROC COMPARE

It is beyond the scope of this paper to delve too deep in to this aspect, but the take home message is that the PROC COMPARE statement of “All values compared are exactly equal” does NOT mean that the TWO data sets are in fact IDENTICAL. To prove two data sets are in fact identical we run PROC COMPARE three times on different aspects of the data.

Query to data dictionary	PROC compares
Compare 1: Compare of the ACTUAL data	<pre>/* Compare 1 ; proc compare base=&baseds (encoding=asciiany) compare=&newds (encoding=asciiany) out = work._compare_&basedsm outnoequal ; run ;</pre>
Compare 2: Compare of the attributes of the variables (name, type, length, label)	<pre>/* Compare 2 ; proc compare base = work._source_vars compare = work._target_vars out = work._compare_structure outnoequal noprint ; run ;</pre>
Compare 3: Compare of the attributes of the data sets (member label, number of observations, number of variables)	<pre>/* Compare 3 ; proc compare base = work._source_dsstruct compare = work._target_dsstruct out = work._compare_dsstruct outnoequal noprint ; run ;</pre>

The misnomer of PROC COMPARE

It is beyond the scope of this paper to delve too deep in to this aspect, but the take home message is that the PROC COMPARE statement of “All values compared are exactly equal” does NOT mean that the TWO data sets are in fact IDENTICAL. To prove two data sets are in fact identical we run PROC COMPARE three times on different aspects of the data.

By running PROC COMPARE three times we can get beyond the inherent feature of PROC COMPARE that only matches like for like commonly named variables, we can demonstrate that the data sets are IDENTICAL. All three PROC COMPARES create an “output dataset”, if all three datasets are EMPTY (contain ZERO observations) then both data sets are “IDENTICAL”. We test for this scenario.

Program and macro files

Within our organization, for SAS files we consider some as “programs” and some as “macros”.

Our differential is:

- SAS files that we may execute with the “sas” command are “program” files
- SAS files that are processed per SAS Autocall are “macro files”

We actually store them in different folders. We have a similar approach for R programs and R functions, etc.

This difference is differentiated in this paper because our “program” files are the files that define and reference the file system, for example file and data libref paths are in the program files, and possibly set in to macro variables.

As a general rule our “macro” files are environment agnostic, these macros could potentially be migrated to any environment and should work “without change”

Program and macro files

Summary of differences

“Program” files	“Macro” files
Executed by the “sas” command Contain statements that define the environment, including file and data libref paths (which may be stored to macro variables)	Invoked by the “program” files by SAS Autocall (we generally do not use %INCLUDE) Do not contain file or data libref paths Potentially portable between environments with no change
As such	
When ported to a new environment, the file and data paths would be expected to change. Therefore, a direct electronic comparison of the “program” files between the source (old) and target (new) system would likely fail and report the files as “different” (changed). It is possible for a given project area, we have 0% of files matched	Are expected to be 100% identical in both environments because they contain ZERO statements that are environment specific

Per files potentially having “known” or “expected” differences

LINUX base tools will report the files as different

- However, we prefer that we understand that the difference(s) are expected per predictable change

We therefore run the files through “sed” first

The “sed” LINUX utility

Per GNU documentation, “sed” is:

- *The sed (Stream Editor) command in Linux is a powerful utility used for processing and manipulating text in files and streams. Unlike traditional text editors, sed operates in a non-interactive manner, meaning you specify the transformations you want to apply ahead of time, and sed executes them without further user interaction.*

We use “sed” in the following way:

1. Prior to running “sed”, we run LINUX diff on the source and target files and capture the return code (0 meaning the files are the same, ≥ 1 they are different)
2. If the files are not identical, we then do the following through step (4). We copy the source and target files to a scratch area
3. We process both files through “sed” using a “sed script” that is aligned to the expected differences between our source and target environments, and makes the text in both files that is “environment specific” -> the same (we use a scratch area because “sed” does modify the files and we MUST NOT change the base files)
4. We then re-run LINUX diff on the files in scratch looking to see if they remain different, or not

Per the above we have TWO return codes:

1. Pre “sed”, are the files different (deduced with the “diff” command)? Yes or No (1 or 0)
2. Post “sed”, are the files different? Yes or No (1 or 0)

Example “sed” file and use

The “sed” script being:

common.sed

```
#Prior system common environment specific values

s#%%include temp#%%include initialization-file#g
s#%%include TEMP#%%include initialization-file#g

#New system common environment specific values

s#%%include "/Volumes/data/newsystem/prod/system_init.sas"#%%include initialization-file#g
```

Simple summary. Each “sed” script command above is in four parts. Each part (note # is the delimiter)

- 1) Is the action, in this case “s” means substitute
- 2) Value to detect
- 3) Value to substitute
- 4) Scope of substitute, “g” mean global (so through the entire file)

For the purpose of getting a “compare match” using the **diff** command, the substitutions all resolve to the “same value”, in this case to: **%%include initialization-file**

Running “sed” on one or more files

The “sed” script per prior slide was “common.sed”

LINUX command

```
#From SAS, execute host commands with the X command  
  
x  "sed -i.bak -f common.sed &base_path/&base_file" ;  
x  "sed -i.bak -f common.sed &comp_path/&compare_file" ;
```

Simple summary. There are TWO sed commands, the first is executed against the file copied from the old (or source) system, and the second one against the file from the new (or target) system.

Both are the same structure, both use the same “sed script”.

- 1) -i means to create a backup copy of the file prior to running “sed”, this maintains the original or pre-change file
- 2) -f is the “sed script” file to execute, in this case “common.sed”
- 3) The remainder of the line is the filename to change, in this example it is resolving SAS macro variables. NOTE: The file(s) to change could include wildcard characters, for example *.sas

After execution, the files will have been in-line edited or changed per the content in this case of “common.sed”

The “sed” LINUX utility with “diff”

The assessments are as follows:

Pre “sed”	Post “sed”	Assessment
Files are identical	N/A (not run)	Files are assessed “identical
Files are different	Files are now the same	All changes are in the “expected list”, the file is assessed as “Different but with permissible changed”
Files are different	File remain different	At least one change was not expected, this is then reported and the programming lead makes a manual determination

From SAS, the LINUX **diff** command is processed as a host command from the SAS macro, and the return code is captured from SYSRC. LINUX diff has the following return codes:

Return code	Meaning
0	(Zero) the files are detected as electronically identical
1 (and in some instances 1+)	The files are detected as different (in some configurations the value may be the count of differences)

The “diff” command

We run it multiple times

diff examples

```
#1 No output, we capture the output through the SYSRC automatic SAS macro variable
x "diff -q -w -t -E -B &base_path/&base_file &comp_path/&compare_file";
%let diff_count1 = &sysrc ;

#2 Spooling the output to a "diff file"
x "diff -w -t -E -B &base_path/&base_file &comp_path/&comp_file      >> &report_path./diff_program_&file..txt";

#3 Spooling the output to a "sdiff file"
x "diff -y -w -t -E -B &base_path/&base_file &comp_path/&comp_file >> &report_path./sdiff_program_&file..txt";
```

- 1) With the -q parameter. This means “quite” and does not spool any output. We detect the return code from this, with 0 meaning the files are identical and 1 (or more) meaning the files are detected different

If differences per (1) are detected, we issue two more diff commands and these spool output that we capture

- 2) With the -w, t, E and B parameters to spool a standard “difference report”
- 3) With the addition of the -y parameter to spool a side by side (or sdiff) report

Parameter files

Our historical system stored parameter files as text based files, an example:

Parmfile1.info

```
title1^This is the first title  
title2^Second title  
foot1^First footnote  
dsname^adam.adsl
```

- We then read the parameter file and store initially as a data set. In our system use, everything to the left of the “^” becomes a global macro variable name, and everything to the right of the “^” is the macro variable value.
- In our new system, these parameters are in JSON files. Probably plainly given the difference in these file types and electronic compare is NOT possible.
- We therefore read the JSON file and instantiate a data set where we build value pairs to mimic the structure of our prior system. We then PROC COMPARE these two data sets spooling an output data set with the expected result that the generated compare output data set is:
 - Empty if all parameters name any parameter values are ALL the same
 - Not empty if there is one or more differences

Example reporting

Each comparisons is recorded, each match is consisted “No differences” and each not match is considered for “permissible differences” or “unexpected difference”.

- These assessments are spooled to a HTML report that we build (append to) during the evaluations. At the end of the processing we read the HTML file in to a SAS data sets, and email it to the programming lead (who is the person that runs the program per study).

HTML headers	HTML footers
<pre>%let saswork = %sysfunc(pathname(work)) ; filename indexhtm "&saswork/index.html" ; data _null_ ; rundatetime = datetime() ; call symput('rundatetime',trim(left(put(rundatetime,datetime20.)))) ; format rundatetime ; run ; data _null_ ; file indexhtm ; rundatetime = "&rundatetime"dt ; put '<html>' ; put '<head>' ; put '<meta content="text/html; charset=ISO-8859-1" http-equiv="content-type">' ; put "<title>Differences report - &base_path to &comp_path</title>" ; put '</head>' ; put '<body bgcolor="#ffffff">' ; ...</pre>	<pre>data _null_ ; file indexhtm MOD ; datetime = datetime() ; format datetime datetime. ; put ' </td></tr>' ; put ' </table>' ; put ' </td></tr>' ; put '</table>' ; put '</div>' ; put "<p>This report was generated by PROGRAM. <small>&sysprocessname by &sysuserid, processing completed at: " datetime datetime. "</small></p>" ; put '</body>' ; put '</html>' ; run ;</pre>

Reading the HTML back and emailing

Abridged code for reading the HTML back and sending an email message. Please see the full paper for expanded example:

Reading the HTML back into a data set	HTML footers
<pre>data work.sendemail ; infile indexhtm truncover ; length text \$1024 ; input text \$1-1024 ; run ;</pre>	<pre>options emailsys=smtp ; filename sendit email "foo" content_type="text/html" ; data _null_ ; set work.sendemail end = eof ; file sendit ; if _n_ = 1 then do ; put "!EM_TO!%cmpres(&email_contact_to)" ; put "!EM_CC!%cmpres(&email_contact_cc)" ; put "!EM_BCC!%cmpres(&email_contact_bcc)" ; put "!EM_FROM!%cmpres(&email_contact_from)" ; put "!EM_REPLY!%cmpres(&email_contact_replyto)" ; put "!EM_SUBJECT!Differences report" ; end; put text ; if eof then do ; put '!EM_SEND!'; put '!EM_NEWMSG!'; put '!EM_ABORT!'; end ; run ;</pre>

Example reporting (from the HTML / email message)

Example for data sets		Example for MACROS	
Dataset	Result	File	Result
SDTM.AE	No change diff comparison data AE.txt	adae.sas VS adae.sas	No change diffs sdiffs
SDTM.CE	No change diff comparison data CE.txt	adae_ns_tier_toc.sas VS adae_ns_tier_toc.sas	No change diffs sdiffs
SDTM.CM	No change diff comparison data CM.txt	adae_protocol_derivations.sas VS adae_protocol_derivations.sas	No change diffs sdiffs
SDTM.CO	No change diff comparison data CO.txt	adae_s_vax_bydict.sas VS adae_s_vax_bydict.sas	No change diffs sdiffs
SDTM.CV	No change diff comparison data CV.txt	adae_s_vax_summary.sas VS adae_s_vax_summary.sas	No change diffs sdiffs
SDTM.DM	No change diff comparison data DM.txt		

Example for PROGRAMS		Example for Parameter Files	
File	Result	File	Result
adcevd.sasdrv VS adcevd.sas	Permissible diffs sdiffs	adce_f001_lr_maxsev.tot VS ADCE_F001_LR_MAXSEV.json	Different Parameter report
adce_s020.sasdrv VS adce_s020.sas	Permissible diffs sdiffs	adce_f001_se_maxsev.tot VS ADCE_F001_SE_MAXSEV.json	Different Parameter report
adfacevd.sasdrv VS adfacevd.sas	Permissible diffs sdiffs	adce_s010_lr_age.tot VS ADCE_S010_LR_AGE.json	Different Parameter report

Conclusions

Through the techniques outlined above we have created a robust solution that we can run on our studies and in moments have a report of the conformance level, including:

- How many (ideally all) of the data sets are the same (and if there are differences, what they are)
- How many of our program files, be that SAS programs, SAS macros (also R code), and other flat files are the same (and where differences exist, what they are)
- How many of our parameter files are the same (and if there are differences, what they are)

Our lead programmers then review these reports and can make a quick decision on the success or not of the migration, and if there are issues, they can quickly focus on what those specific issues are.

Thank You

Kishan Korsapati

Email: PadmaKishanReddy.Korsapati@pfizer.com

Darrell Edgley

Email: Darrell.Edgley@pfizer.com

Company: Pfizer Inc

Address: 500 Arcola Road, Collegeville, PA 19426, USA

Website: <http://pfizer.com>

Your comments and questions are valued and encouraged.

Thank you.

