Paper SM09

Automated File Comparison and Validation Across Programming Platforms

Padma Kishan Korsapati, Darrell Edgley,

Statistical Data Sciences & Analytics (SDSA), Pfizer Inc, Collegeville, PA, USA

ABSTRACT

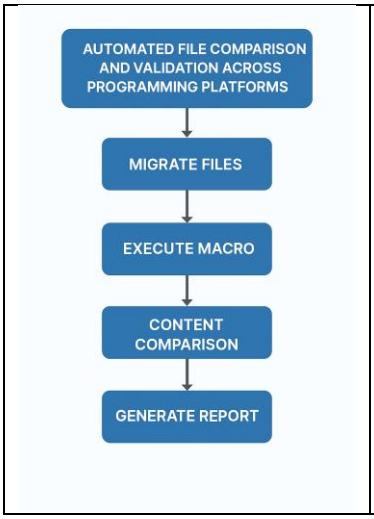
Pharmaceutical companies periodically upgrade their integrated development environments (IDEs), requiring migration of code, macros, and data sets from older to newer systems. Additionally, legacy studies may be archived to free up storage for ongoing work. These transitions often involve bulk copying of files, which can be error-prone and time-consuming if validated manually. To address this, we developed a macro in SAS that automates the comparison and validation of files across programming environments. It ensures all files and macros are successfully transferred and checks for content consistency. For data sets, the macro uses PROC COMPARE to summarize differences, while discrepancies in other file types are highlighted in a text report. This approach significantly reduces manual effort, improves accuracy, and supports efficient migration and archival processes in regulated environments.

INTRODUCTION

It is of paramount importance that when we upgrade/migrate between environments, we do so with accuracy such that we can quickly verify, even prove that the upgrade/migration is successful and complete. This may include system level code and data, plus study level code and data. The “data” may also be more than “study/clinical data”; it may also include metadata and parameter files. Our operating environment is LINUX based. The examples hereinafter are therefore LINUX (UNIX) focused, but the principles could be applied to any operating system through use of the appropriate tools.

This diagram is a simplistic overview of the situation we are solving. Separately, we have built migration tools that are used for the “lift and shift” from the legacy environment into the new or target environment.

From the diagram below, this paper focuses on the “execute macro”, “content comparison” and “report generation”.



We called our macro **%check_for_migration**. It takes multiple parameters and actually has 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 whether "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.

The Pfizer configuration for our programming teams in "study support"

In simple terms within our organization, we have main project folders, then a standard set of sub-folders below that folder. For example, we have a structure such as:

/home/system_name_one/projects/proj1

Which hereinafter I will refer to as the "base project area", and then below that folders for the types of files within or created, for example:

./adam/

./analysis

./logs

./macros/

./programs

./sdtm

./tlfs

./tools

The macro

Written in SAS. Uses SAS/Autocall. If not familiar with AUTOCALL please refer to the paper "Autocall Macro Libraries"[1]. Statements such as %include are not part of this code solution.

Key parameters:

Parameter	Description
BASEPROJ	The base project folder of our historical reporting system
TARGETPOJ	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 are 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

Our utility macro verifies all components of the study migration from our legacy system to our new system. These components include:

- SDTM data sets
- ADaM data sets
- QC and PRODUCTION code
- Macros (within our organization our codebase is highly macrotized). Our more recent codebase is also using a lot of R and stored R-functions
- Parameter files (our legacy system uses text-based file with "parameter pairs", our new system stores the parameters in "JSON files")

Further, for reporting, our utility goes deeper than detecting that any individual component "is different". It also provides a report with links so that the differences can be quickly viewed and assessed.

TYPES OF CHECKS

SAS data sets

The base check on SAS data sets is using PROC COMPARE. However, in using PROC COMPARE we do take into consideration "The Misnomer of PROC COMPARE"[2] which in simple terms means in running PROC COMPARE do not consider two data sets identical

just because you get the procedure compare output statement of “No unequal values were found. All values compared are exactly equal”.

In evaluation of data sets being equal, we perform PROC COMPARE three times on each data set, with each PROC COMPARE producing an output data set. The PROC COMPARE’s executed are:

1) Compare of the actual data. This is the typical use case for PROC COMPARE. This also spools output which is captured by ODS LISTING
2) Compare of the data set variable structure, generated with a query to the DICTIONARY.COLUMNS data dictionary view
3) Compare of the “data set” key attributes, generated with a query to the DICTIONARY.TABLES data dictionary view

We feel it is prudent to run the three compares per the known “misnomers” in PROC COMPARE. The above three, in order, verify:

1) For matching variables that have the same variable type, we have a full match on data values
2) Dataset variables are identical. Per PROC COMPARE not comparing new variables in the target data set, or omitted deleted variables from the source data set, or if although unexpected a variable type was changed from CHAR to NUM (or visa-versa), this compare will tell us that
3) We do not have additional records. Per PROC COMPARE not comparing additional records, for example the base data set contain 100 observations, and the next data set also contained the same 100 observations but then an appended set of 10, PROC COMPARE would still report “All values compared equal”

Example code:

Query to data dictionary	PROC compares

`%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`
	`%* 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 ;`

With this structure of PROC COMPARE, the expected output for an "identical data set" in both environments will be for the three WORK data sets to ALL be "empty", meaning that they contain ZERO observations.

Our code solutions verify all three contain ZERO observations. If true, the data set is reported as "identical"; if any data sets does NOT contain ZERO observations, the data sets are reported as "different".

Please see the **APPENDIX** for code to deal with the reporting.

SAS program and macro files

Within our organization within SAS, we parlance SAS files in two ways. We consider "SAS programs" the file that we run from the command line with the "sas" command, and we consider the macros that get called using "AUTOCALL" the SAS macros. We have a similar opinion with R, with the main "R program" and the "R functions" we have deployed.

In general terms, we expect code (in our organization) that interfaces with the file system to define those interactions within the "SAS programs", and typically the "SAS macros" will NOT do this.

If "sas macros" need to reference the file system, they typically will use items created by the "sas program", for example a macro variable for the file path, etc. Per this, typically we expect that the vast majority of "sas macros" from our source system will be 100%

identical in the target system, but it is also quite possible that 0% of our "sas programs" from our source system match the "sas programs" in the target system.

Especially with "sas programs", these differences create a challenge with comparing the files and ensuring that the files are consistent. However, we do know the structure of our source system, and the structure of our new (target) system, and these have predictable code changes.

The LINUX tool: "sed"

"sed"[3] is the "LINUX" (or UNIX) "stream editor". Brief summary of "sed":

> *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? Yes or No (1 or 0)
2) Post "sed", are the files different? Yes or No (1 or 0)

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"

Pre "sed"	Post "sed"	Assessment
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)

Our code solution essentially treats the return code as Boolean, with ZERO interpreted as the files are identical and >= 1 as the files are not the same. We do run the "diff" command multiple times with different options. Please refer to the diff[4] manual for the meaning of all the options. A brief summary of the parameters however includes we run in QUIET mode (initially not spooling any output just detecting the return code for differences), ignoring differences that are per WHITE space, TABS/spaces differences etc).

NOTE: In SAS the "x" command is the method for issuing a "host command", the command issue is interpreted by the "host operating system". SAS will detect the host command "return code".

Base use of "diff" and capturing the return code:

```
x "diff -q -w -t -E -B &base_path/&base_file &comp_path/&compare_file";
%let diff_count1 = &sysrc ;
```

If differences are found:

```
%if &diff_count1 NE 0 %then %do ;
  %put %str(IN)FO: Differences were found ;
  %************************************************************************** ;
  %* Per the DIFF command the files have been assessed as different          ** ;
  %* (not identical)...                                                      ** ;
  %************************************************************************** ;

  %************************************************************************** ;
  %* Run the files through "sed"                                             ** ;
  %************************************************************************** ;

  x   "sed -i.bak -f common.sed &base_path/&base_file" ;
  x   "sed -i.bak -f common.sed &comp_path/&compare_file" ;

  x "diff -q -w -t -E -B &base_path/&base_file &comp_path/&compare_file";
  %let diff_count2 = &sysrc ;

  %if &diff_count2 NE 0 %then %do ;
    %put Files remain different post sed ;
```

If differences are found:

```
  %end ;
  %else %do ;
    %put Files are now the same post sed, so all changes are permissible ;
  %end ;
%end ;
```

Please see the **APPENDIX** for creation of the differences report, and for the "common.sed" script file.

Parameter files

In our historical system our parameter files are text based with value pairs, for 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 (see the code for data set compare) 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

CONCLUSION

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.

REFERENCES

#	Reference
1	MWSUG Paper 143. Autocall Macro Libraries, by Ken Schmidt https://www.lexjansen.com/mwsug/2012/BI/MWSUG-2012-BI18.pdf
2	"The Misnomer of PROC COMPARE" by Alex Ostrowski, Pfizer, PharmaSUG 2020
3	LINUX "sed" https://www.gnu.org/software/sed/manual/sed.html
4	LINUX "diff" https://www.gnu.org/software/diffutils/

ACKNOWLEDGEMENTS

The authors would like to acknowledge the assistance of our SDSA colleagues and Standards Team.

CONTACT INFORMATION

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

Author Name: Padma Kishan Korsapati
Company: Pfizer Inc
Address: 500 Arcola Road, Collegeville, PA 19426, USA
Email: PadmaKishanReddy.Korsapati@pfizer.com

Author Name: Darrell Edgley
Company: Pfizer Inc
Address: 500 Arcola Road, Collegeville, PA 19426, USA
Email: Darrell.Edgley@pfizer.com
Website: http://pfizer.com

APPENDIX

General reporting technical approach

General reporting of the exceptions consists of two parts:

1) Spooling of an output file showing the differences

Depending on the type of comparison/difference check the method of creating this spooled file differs

2) Augmenting a running differences report

The main difference report is an HTML file. This is incepted with HTML headers at the start of the macro utility execution and closed with HTML footers towards the end of the macro utility execution.

HTML Headers

```
%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 ;
  format rundatetime datetime. ;

  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">' ;
  put '<div align="center">' ;
  put '<table border="1"' ;
  put ' <tr><td>' ;
  put ' <table border="0" width="800pt">' ;
  put "  <tr><td bgcolor='green'><font color='white'>Differences Report</font></td></tr>" ;
  put '  <tr><td>' ;
  put "    <b>This report was generated using data that existed as of " rundatetime " and its
contents reflects files found at that instant in time.</b>" ;
  put '    </td></tr>' ;
  put '    <tr><td>' ;
run ;
```

HTML Footers

```
data _null_ ;
  file indexhtm MOD ;
  datetime = datetime() ;
  format datetime datetime. ;

  put '    </td></tr>' ;
  put '  </table' ;
  put '  </td></tr>' ;
```

HTML Footers

```
  put '</table>' ;
  put '</div>' ;
  put "<p>This report was generated by PROGRAM. <small>(<b>&sysprocessname</b> by
<b>&sysuserid</b>, processing completed at: <b>" datetime datetime. "</b>)</small></p>" ;
  put '</body>' ;
  put '</html>' ;
run ;
```

We also email the HTML report. Our email system supports HTML formatted email. Base SAS code for sending HTML email is:

Read the generated HTML back into SAS

```
data work.sendemail ;
  infile indexhtm truncover ;
  length text $1024 ;
  input text $1-1024 ;
run ;
```

SAS code for sending an email (abridged)

```
options emailsys=smtp ;
filename sendit email "foo" content_type="text/html" ;

data _null_ ;
  %************************************************************************* ;
  %* Actually send the email message...                                    ** ;
  %************************************************************************* ;
  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 ;
    %********************************************************************** ;
    %* Example of including an attachment. This example is an Excel     ** ;
    %* spreadsheet.                                                       ** ;
    %********************************************************************** ;
    %if %mu_fileexist(%str(&outfile_main_full..xls),alert=i) %then %do ;
      put '!EM_ATTACH!("' "%str(&outfile_main_full..xls)" '"
                    content_type="application/excel")' ;
    %end ;

    put '!EM_SEND!';
    put '!EM_NEWMSG!';
    put '!EM_ABORT!';
  end ;
run ;
```

common.sed

An example “sed” script for modification of program files (we modify them in a copied to “SCRATCH area”):

```
#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
```

In this example the delimiter is the # character. A common delimiter in documentation is the / character but we can not use this as the “/” features in the values we are searching for.

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” means 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**

Reporting of SAS differences

The SAS differences are detected as outlined above by running three instances of PROC COMPARE on different sets of the data set (i.e. the verbatim data, the attributes of the variables and the attributes of the data set that includes the observation count).

If differences are detected the differences report is spooled by encapsulating the PROC COMPARE execution within ODS LISTING blocks, and then adding a link to the spooled report.

Encapsulated PROC COMPARE
(note processed in a loop over all data sets in the data library)

```
%put %str(IN)FO: &sysmacroname: Loop &i of %trim(&cnt), spooling DATA COMPARE to:
&indexp./diff_&out._&dat..txt ;
ods listing file= "&report_path/diff_dataset._&dat..txt";

%*************************************************************** ;
%* Compare the data sets which will create a report in the redirected ** ;
%* LISTING file plus also a compare output data set.  If this data    ** ;
%* set is EMPTY then no data differences were found.                  ** ;
```

Encapsulated PROC COMPARE
(note processed in a loop over all data sets in the data library)

```
%************************************************************************* ;
proc compare base    = b_lib.&dat (encoding = asciiany)
             compare = t_sib.&dat (encoding = asciiany)
            out      = work._compare_&i (compress = yes) outnoequal ;
        run;

%let diff=&sysinfo;
%let diff_count = %util_num_obs(_dsn=work._compare_&i) ;

%************************************************************************* ;
%* Close the LISTING output file...                                     ** ;
%************************************************************************* ;
ods listing close;
```

Augmenting the HTML report
Also part of the same loop

```
data _null_ ;
  file indexhtm MOD ;

  put "<tr>" @ ;
  put "<td>&dat</td>" ;

  put "<td>" @ ;

  %if %eval(&diff_count) = 0 %then %do ;
    put "<font color='green'>No change</font> " @ ;
  %end ;
  %else %do ;
    put "<font color='red'><b>Different</b></font> " @ ;
  %end ;

  put "<a href='" "&report_path/diff_dataset._&dat..txt" "'>diff_dataset_&dat..txt</a>" @ ;
  put "</td>" ;
  put "</tr>" ;
run ;
```

Example HTML report snip

Dataset	Result
SDTM.AE	No change diff_comparison_data_AE.txt
SDTM.CE	No change diff_comparison_data_CE.txt
SDTM.CM	No change diff_comparison_data_CM.txt
SDTM.CO	No change diff_comparison_data_CO.txt
SDTM.CV	No change diff_comparison_data_CV.txt
SDTM.DM	No change diff_comparison_data_DM.txt

Reporting of flat file differences (including SAS programs/macros)

As outlined above, comparison of flat files which includes SAS programs and macros, R files etc is performed by running the LINUX **diff** command with the **-q** parameter. This means QUIET mode and it only detects if the file is different or NOT, it does not actually detail the differences.

On detecting that a file is different a reporting section is invoked. This then runs **diff** twice more, both times producing out. The first output file is a typical "diff" file where the differences are listed in a linear fashion, the filename prefix is "diff". The second output file is a side by side "diff" file which is also the format generated by the "sdiff" command. In our implementation we use the **diff** command but force output to be compatible with **sdiff**. The spooled file has a filename prefix "sdiff".

Spooling the DIFF files

```
x "diff    -w -t -E -B &base_path/&base_file &comp_path/&comp_file >>
                                          &report_path./diff_program_&file..txt ";
x "diff -y -w -t -E -B &base_path/&base_file &comp_path/&comp_file >>
                                          &report_path./sdiff_program_&file..txt";
```

The augmentation to the HTML report is very similar to the method for SAS data sets. However, we expect data sets to be identical, for flat files in our environment we expect some may have expected changes, as such we have a SECOND **diff** return code after the **sed** command has been processed. If a file is different but then compares identical after **sed** then we consider the changes "permissible", this is handled in code as follows:

Augmenting the HTML report

```
data _null_ ;
  file indexhtm MOD ;

  put "<tr>" @ ;
  put "<td>&typ..&dat</td>" ;

  put "<td>" @ ;

  %if %eval(&diff_count) = 0 %then %do ;
    put "<font color='green'>No change</font> " @ ;
  %end ;
  %else %do ;
    %if %eval(&diff_count2) = 0 %then %do ;
      %************************************************************* ;
      %* If COUNT_DIFF2 is ZERO it means that the file is DIFFERENT   ** ;
      %* but the SECOND check for differences is ZERO.  If this is so ** ;
      %* this is AFTER the processing of the sed script that          ** ;
      %* eliminates differences common to CDARS and SIGMA.  i.e.      ** ;
      %* apart from expected CDARS to SIGMA differences the files     ** ;
      %* are identical.  Files where this is true will be considered  ** ;
      %* PERMISSIBLE for differences whereas files where this is      ** ;
      %* NOT true are considered UNEXPECTED for differences.          ** ;
      %************************************************************* ;
      put "<font color='orange'><b>Permissible</b></font> " @ ;
    %end ;
    %else %do ;
      put "<font color='red'><b>Different</b></font> " @ ;
```

Augmenting the HTML report

```
    %end ;
  %end ;

  put "<a href='" "&report_path/diff_program_&file..txt" "'>diffs</a>
       <a href='" "&report_path/sdiff_program_&file..txt" "'>sdiffs</a>" ;

  put "</td>" ;
  put "</tr>" ;
run ;
```

Example HTML report snip

Macros (in our environment we do not expect these to change):

File	Result
adae.sas VS adae.sas	No change diffs sdiffs
adae_ns_tier_toc.sas VS adae_ns_tier_toc.sas	No change diffs sdiffs
adae_protocol_derivations.sas VS adae_protocol_derivations.sas	No change diffs sdiffs
adae_s_vax_bydict.sas VS adae_s_vax_bydict.sas	No change diffs sdiffs
adae_s_vax_summary.sas VS adae_s_vax_summary.sas	No change diffs sdiffs

Programs (the SAS programs that are invoked from the command line), we do expect these to change but typically with expected changes:

File	Result
adcevd.sasdrvr VS adcevd.sas	Permissible diffs sdiffs
adce_s020.sasdrvr VS adce_s020.sas	Permissible diffs sdiffs
adfacevd.sasdrvr VS adfacevd.sas	Permissible diffs sdiffs

Parameter files

In our old system these are text based value pair file, in our new system they are JSON files. We read both sets in to similarly structed SAS data sets and then run PROC COMPARE on them. The method of spooling and HTML file augmentation is very similar to the other file types and data set.

Example HTML report snip

File	Result
adce_f001_lr_maxsev.tot VS ADCE_F001_LR_MAXSEV.json	**Different** Parameter report
adce_f001_se_maxsev.tot VS ADCE_F001_SE_MAXSEV.json	**Different** Parameter report
adce_s010_lr_age.tot VS ADCE_S010_LR_AGE.json	**Different** Parameter report