PharmaSUG 2016 - Paper BB04

Go Compare: Flagging up some underused options in PROC COMPARE

Michael Auld, Ampersoft Ltd, London, UK

A brief overview of the features of PROC COMPARE, together with a demonstration of a practical example of this procedure's output data set. Identify observations that have changed during the lifetime of a study when different cuts have been made to varying snapshots of the same database, then create flags from this metadata that can be used in safety update CSR listings.

INTRODUCTION

Most of us have used PROC COMPARE, one of the most useful and essential procedures that help make SAS® standout from the other statistical software packages out there. At its heart is the simple ability to show the differences between two superficially identical data sets. In the pharmaceutical world, where there is a regulatory obligation to get things right, the procedure has come into its own where SOPs demand double-programming of data.

But how much of this procedure do you really know? This paper will attempt to refresh your knowledge on some of the lesser used and lesser understood pieces of syntax before demonstrating a practical use of the output data sets generated from the procedure. I don't wish to agonizingly regurgitate all of the SAS Procedures guide, but focus on one or two handy options, what they mean and why I personally I have found them useful in my experience.

SYNTAX

The basic syntax of the procedure can be summarized as follows:

```
PROC COMPARE <options>;
BY <sortoptions> variables;
ID <sortoptions> variables;
VAR variables;
WITH variables;
RUN;
```

with DATA= and COMPARE= reflecting the two data sets that you wish to compare. NB DATA= can be substituted with BASE= as they perform the same function – that of identifying the BASE data set to be compared with the COMPARE data set.

TYPICAL OUTPUT FROM PROC COMPARE

PROC COMPARE delivers its results in a very structured way. By default, you should expect to see the following:

- *DATASET SUMMARY*
  A description of the data set(s) that are being compared detailing their data set labels, number of observations and variables, and creation dates and times are shown here.
- *VARIABLE SUMMARY*
  This section summarises any differences (in the attributes) of the variables in the data sets.
- *OBSERVATION SUMMARY*
  This section summarises how many observations vary between the compared data sets.
- *DIFFERENCES SUMMARY*
  Probably the most interesting section of the report, showing the values of the variables in your data sets that differ.
- *OBSERVATION AND VARIABLE COMPARISONS*
  You may have some observations or variables that only appear in one of the data sets exclusively. These will be listed here

```
                                The COMPARE Procedure
                            Comparison of ADSDEV.ADCM with ADS.ADCM
                                      (Method=EXACT)

                                    Data Set Summary

Dataset              Created          Modified  NVar    NObs  Label

ADSDEV.ADCM  15APR10:09:39:14  15APR10:09:39:14    39     564  Concomitant Medications
ADS.ADCM     15APR10:15:11:48  15APR10:15:11:48    39     564
```

Figure 1. Data set summary

```
                                    Variables Summary

                          Number of Variables in Common: 39.
                          Number of Variables with Differing Attributes: 13.


                        Listing of Common Variables with Differing Attributes

Variable  Dataset      Type  Length  Format  Informat  Label

                       Char      15  $15.              Study Identifier
          ADS.ADCM     Char      15  $15.    $12.      Study Identifier
```

Figure 2. Variable summary

```
                                    Observation Summary

                                Observation      Base  Compare

                                First Obs           1        1
                                First Unequal     508      508
                                Last  Unequal     558      558
                                Last  Obs         564      564

                    Number of Observations in Common: 564.
                    Total Number of Observations Read from ADSDEV.ADCM: 564.
                    Total Number of Observations Read from ADS.ADCM: 564.

                    Number of Observations with Some Compared Variables Unequal: 4.
                    Number of Observations with All Compared Variables Equal: 560.
```

Figure 3. Observation summary

```
                                Variables with Unequal Values

Variable  Type  Len   Label                    Ndif   MaxDif  MissDif

CMTYPE    CHAR    1   Medication Type             2                0
CMTYPEN   NUM     8   Medication Type, Numeric    2    1.000       0
CMDOSE    CHAR  200   Dose per Administration     1                0
CMOBSFL   CHAR    2   Changes to Listing          1                1


                        Value Comparison Results for Variables

                                ||  Medication Type
                                ||  Base Value          Compare Value
                           Obs  ||  CMTYPE               CMTYPE
                          _____ ||  _                    _
                                ||
                            508 ||  P                    C
                            535 ||  P                    C
```

Figure 4. Differences summary

```
                                The COMPARE Procedure
                            Comparison of WORK.MYCM with SDSOLD.CM
                                        (Method=EXACT)

                              Comparison Results for Observations

Observation 22697 in WORK.MYCM not found in SDSOLD.CM: USUBJID=0086-0009
 CMSEQ=49.

Observation 22925 in WORK.MYCM not found in SDSOLD.CM: USUBJID=0088-0004
 CMSEQ=17.

Observation 22953 in WORK.MYCM not found in SDSOLD.CM: USUBJID=0088-0009
 CMSEQ=18.

Observation 22961 in WORK.MYCM not found in SDSOLD.CM: USUBJID=0088-0010
```

Figure 5. Observations and variable comparisons

COMPARE AND ODS TABLES

Just as in other SAS procedures, if you wrap the call to COMPARE with

```
ODS TRACE ON;
```

(and the corresponding ODS TRACE OFF), you can get to see the ODS data sets that SAS uses internally to produce the output from the procedure.

A list of the ODS table names (together with the section that is produces) are illustrated below:

Section of COMPARE output	ODS Table Name
Data Set Summary	CompareDatasets
Variable Summary	CompareVariables
Observation Summary	CompareSummary
Differences Summary	CompareDifferences
Observation and Variable Comparisons	CompareDetails

Table 1. Compare output and ODS table name equivalents

It is worth mentioning that the ODS data sets are only created during the production of the report, so if you are lucky to have an exact match of your data with the message:

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

Or unlucky for example, when using an ID statement and your data sets are not sorted – then one or more of these sections of the report will not generated and the data sets not produced.

Each of the sections of the report are in fact optional! So, you can control the amount of output generated by COMPARE by selecting the relevant ODS data set before submitting your PROC COMPARE statement,

eg

```
ODS SELECT CompareDifferences;
PROC COMPARE
```

etc

STATEMENTS

Most commonly in the pharmaceutical industry, COMPARE is used in tandem with PROC CONTENTS when performing double-programming (QC-ing or validating a SAS data set), or accepting a new version of a delivered database. CONTENTS delivers a report on the metadata, while COMPARE reports on differences in the data observations themselves.

When tracking down these differences in a double-programming environment, it is useful to use the report from COMPARE to drill into your data set to investigate the cause of the difference(s).

ID STATEMENT

For this reason, I find the ID statement of PROC COMPARE very useful. Without it, the report is delivered by observation number only – which makes it slightly more awkward to then drill and subset the data set (particularly if the number of observations are very large.

	Medication Type, Numeric			
	Base	Compare		
Obs	CMTYPEN	CMTYPEN	Diff.	% Diff
508	1.0000	2.0000	1.0000	100.0000
535	1.0000	2.0000	1.0000	100.0000

Figure 6. COMPARE with default behaviour

Secondly, and with particular reference to comparing the data against a previous snapshot of the database, the observation numbers between the data sets may differ for the otherwise exact same data, so a report by observation number would not be useful in that situation!

	Medication Type, Numeric			
	Base	Compare		
SUBJID	CMTYPEN	CMTYPEN	Diff.	% Diff
00010004	1.0000	2.0000	1.0000	100.0000
00010008	1.0000	2.0000	1.0000	100.0000

Figure 7. COMPARE with ID statement used

The format of the ID statement is

```
ID <sortoptions> variables;
```

Sort options should reflect how the data sets have been sorted before invoking the COMPARE, so if one or more variables have been sorted in descending order, the use of the keyword DESCENDING (paired with each relevant variable affected) should be used. Alternatively, if the data has not been sorted (and you still wish to display the results with an ID variable) then use the NOTSORTED option; this only needs to be written once in the ID statement. Again, if the data is not sorted and the ID statement is used without the NOTSORTED option, then you will receive plenty of complaints from SAS in your log and in your output.

BY STATEMENT

This statement behaves much as it does inside other procedures, structuring the output in order of the variable named in the BY statement. Just as for the ID statement, the sort options should be applied in the BY statement as they apply in the data set(s), otherwise SAS will throw an error.

VAR STATEMENT

Here PROC COMPARE reports the differences exclusively only for the variables listed in the VAR statement. If the VAR statement is omitted, then the default behaviour of displaying the differences for all of the variables in the data set applies.

```
PROC COMPARE BASE=sds.ex COMPARE=sdsval.ex;
ID subjid;
VAR visitnum;
RUN;
```

WITH STATEMENT: COMPARING VARIABLES WITH DIFFERENT NAMES IN TWO DATA SETS

Consider the following code which compares the variable VISITNUM in the SDTM with AVISITN in the ADaM:

```
PROC COMPARE DATA=sds.ex COMPARE=ads.adex(RENAME=(avisitn=visitnum));
ID subjid;
VAR visitnum;
RUN;
```

The rename could be avoided by using the WITH statement, which allows you to compare variables that have different names:

```
PROC COMPARE DATA=sds.ex COMPARE=ads.adex;
ID subjid;
VAR visitnum;
WITH avisitn;
RUN;
```

will generate:

Value Comparison Results for Variables

		Visit Number			
		Analysis Timepoint Number			
		Base	Compare		
SUBJID		VISITNUM	AVISITN	Diff.	% Diff
00010004		1.01	0	-1.0100	-100.0000
00010004		-1.00	0	1.0000	-100.0000
00010004		-1.00	0	1.0000	-100.0000
00010004		-1.00	0	1.0000	-100.0000

Figure 8. Output from VAR and WITH statements

There is a 1-2-1 correspondence of the variables in listed in VAR to the variables listed in WITH. The only caution is that If the number listed in the WITH statement is greater than in the VAR statement, then these extra ones will be ignored by COMPARE. In the example below, AVAL will be ignored.

```
PROC COMPARE DATA=sds.ex COMPARE=ads.adex;
ID subjid;
VAR visitnum visit exseq ;
WITH avisitn avisit srcseq aval;
RUN;
```

However, if you list more variables in the VAR than in WITH, then the normal behaviour of VAR will happen with the extra variables, and COMPARE will assume that the variable names for these extra variables are the same for both data sets:

```
PROC COMPARE DATA=sds.ex COMPARE=ads.adex;
ID subjid;
VAR visitnum visit exseq extrt;
WITH avisitn avisit srcseq;
RUN;
```

(VISITNUM will be compared with AVISITN, VISIT with AVISIT, EXSEQ with SRCSEQ, and EXTRT will be compared with EXTRT in both data sets).

WITH STATEMENT: COMPARING VARIABLES WITH DIFFERENT NAMES IN THE SAME DATA SET

WITH is also very powerful as it can allow you to check internal consistency across variables in the SAME data set. So, the following code can check the derived analysis versions of the visit number with the version mapped over from the SDTM version:

```
PROC COMPARE DATA=ads.adex;
ID subjid;
VAR visitnum;
WITH avisitn;
RUN;
```

The report is just like the earlier example, but with a slightly different summary reflecting the label difference:

Variable	Type	Len	Compare	Len	Label	Compare Label	Ndif	MaxDif
VISITNUM	NUM	8	AVISITN	8	Visit Number	Analysis Timepoint Number	78682	8.000

Figure 9. Output from VAR and WITH statements used in same dataset

SOME USEFUL OPTIONS (1)

TRANSPOSE

For a different way of looking at your data, the TRANSPOSE option may appeal. Rather than listing differences by grouping by variable, differences for each individual observation (or ID variable) are listed:

Comparison Results for Observations

SUBJID=00010004:

Variable	With	Base Value	Compare	Diff.	% Diff
VISITNUM	AVISITN	99.00	91.000000	-8.000000	-8.080808

SUBJID=00010004:

Variable	With	Base Value	Compare	Diff.	% Diff
VISITNUM	AVISITN	99.00	91.000000	-8.000000	-8.080808

SUBJID=00010004:

Variable	With	Base Value	Compare	Diff.	% Diff
VISITNUM	AVISITN	99.00	91.000000	-8.000000	-8.080808

Figure 10. TRANSPOSE option output (with an ID statement)

MAXPRINT

DEFAULT:
`MAXPRINT=(500, 50)`

To prevent pages and pages of listing of differences in the data, SAS by default restricts the number of lines of output to 50 lines per variable, and 500 lines overall. For our industry where perhaps the important difference (perhaps even the causal difference for the others in the output) may lurk deeper in the data than 10 pages in. To override this

default value, use MAXPRINT=32767 (which is the highest value that it can be set to) and this should be enough for most data comparisons. If you still wish to restrict the number of lines in your compare output, you can control both the lines per variable and the maximum number of lines overall by placing these limits in brackets

CRITERION

DEFAULT:
`CRITERION=0.00001`

This option reflects the level of precision applied to the data when looking for matches. The need to verify and reproduce results in the pharmaceutical industry is of primary importance, and so this option is to be used with great caution (particularly with parameter-based data). However, as computers are prone to errors when representing floating point numbers and arithmetic, you may find that COMPARE detects differences of the magnitude of 10^-9 occasionally, which may not reflect the precision in which the data was collected. This then becomes a very useful function.

LIST(X) OPTIONS

I have put the X wild card character here to reflect all of the options that allow you to control the observations and variables that are unique in either data set. It is easier to explain what they do in the following table:

Option name	Function
`LISTOBS`	Lists the **observations** that appear exclusively in either data set
`LISTBASEOBS`	Lists **observations** appearing exclusively in the **BASE** data set
`LISTCOMPOBS`	Lists **observations** appearing exclusively in the **COMPAR**E data set
`LISTVAR`	Lists the **variables** that appear exclusively in either data set
`LISTBASEVAR`	Lists **variables** appearing exclusively in the **BASE** data set
`LISTCOMPVAR`	Lists **variables** appearing exclusively in the **COMPAR**E data set
`LISTALL`	Equivalent of issuing **LISTOBS** and **LISTVAR** options

Table 2. Summary of LIST(x) options and their behaviour

COMPARING COMPARE

it is worth mentioning the similarities and the differences between PROC COMPARE and the other procedures available in SAS.

SORT OR INDEX?

As mentioned earlier, the ID statement requires a sorted data set – or the NOTSORTED option applied in the ID statement. This is less true of the BY statement as it is possible for the COMPARE to proceed with an unsorted data set if an index is applied. However, an index on an ID statement has no effect, and the predicted ERROR to the log will still occur.

HANDLING OF FORMATS

Unlike PROC MEANS/SUMMARY, formats in the data take no effect – so the same formatted value (if the original unformatted source data are indeed different values) will not be treated equally or aggregated. Additionally, the output that shows the differences will display the unformatted values in the report – which may be difficult to interpret (especially dates or complex code lists).

OUT= AND THE OUTPUT DATA SET OPTIONS

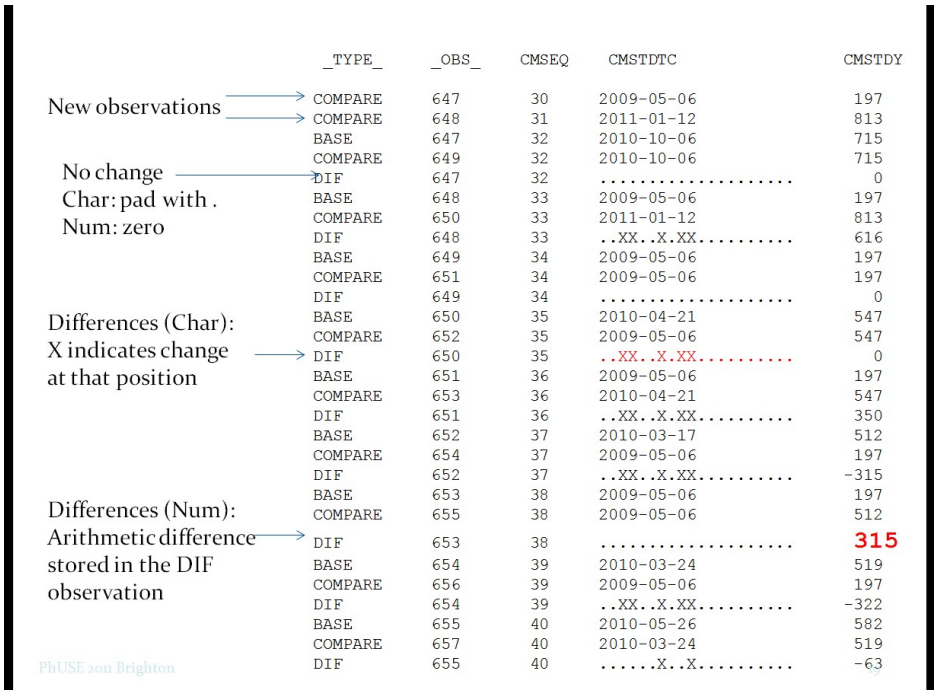
Just as per other SAS procedures, COMPARE has an option to produce and output data set. However, you can control the observations that are written to this data set in COMPARE by using a number of useful OUTxx options. Exercise caution as the size of the output data set may multiply considerably especially if the original data set is already quite large. The options and what they do to the output data set are illustrated in the following table:

Option name	Function
OUTBASE	Outputs **every** observation that appears in the **BASE** data set
OUTCOMP	Outputs **every** observation that appears in the **COMPARE** data set
OUTDIF	Outputs an observation that reflects the differences between the variables in BASE and COMPARE. An observation is still written to the output data set even if no differences are found with the 2 data sets.
OUTPERCENT	Like OUTDIF – but % difference
OUTALL	Equivalent of applying all of the above OUTxx options
OUTNOEQUAL	**Only** writes the observation to the output data set when a difference is detected.

Table 3. Output dataset options

The output data set is written with the names of the observations as in the original data sets, and with the following procedure generated variables: _TYPE_ and _OBS_. The following is a sample output data set that reflects the OUTBASE, OUTCOMP and OUTDIF options. The _TYPE_ column shows the origin of the data (for the BASE and COMPARE data sets) and the _OBS_ the observation number from each data set.

For _TYPE_="DIF", the character variables are populated with an X for each character that is different in the 2 data sets. Masking is done with a "." character. For numeric variables, the value stored is the arithmetic difference between the two values in BASE and COMPARE.



TYPE	_OBS_	CMSEQ	CMSTDTC	CMSTDY
COMPARE	647	30	2009-05-06	197
COMPARE	648	31	2011-01-12	813
BASE	647	32	2010-10-06	715
COMPARE	649	32	2010-10-06	715
DIF	647	32	.....	0
BASE	648	33	2009-05-06	197
COMPARE	650	33	2011-01-12	813
DIF	648	33	..XX..X.XX.....	616
BASE	649	34	2009-05-06	197
COMPARE	651	34	2009-05-06	197
DIF	649	34	.....	0
BASE	650	35	2010-04-21	547
COMPARE	652	35	2009-05-06	547
DIF	650	35	..XX..X.XX.....	0
BASE	651	36	2009-05-06	197
COMPARE	653	36	2010-04-21	547
DIF	651	36	..XX..X.XX.....	350
BASE	652	37	2010-03-17	512
COMPARE	654	37	2009-05-06	197
DIF	652	37	..XX..X.XX.....	-315
BASE	653	38	2009-05-06	197
COMPARE	655	38	2009-05-06	512
DIF	653	38	.....	315
BASE	654	39	2010-03-24	519
COMPARE	656	39	2009-05-06	197
DIF	654	39	..XX..X.XX.....	-322
BASE	655	40	2010-05-26	582
COMPARE	657	40	2010-03-24	519
DIF	655	40	.....X..X.....	-63

Figure 11. Output from VAR and WITH statements

A PRACTICAL APPLICATION

ADDING FLAGS TO DATASETS TO IDENTIFYING CHANGES BETWEEN DATA SNAPSHOTS AND DATA CUTS

By using the output dataset capability of the COMPARE procedure, I was able to detect changes between snapshots of a locked, reported study and an extension study (with the same database). By reading data from dictionary.columns, and looking at DIF, BASE and COMPARE for the same unique keys for each dataset, a report could be sent out to data management reporting on these changes. I expanded on this idea when I was involved in the reporting of a number of long-term oncology studies. The CSRs had already been produced, but now as part of a wider submission, the regulatory authorities had asked for an updated report of the safety data, 120 days following the original reporting. As the same database was being used, it was important to know that the integrity of the original reported data was sound, but if there were any differences these should be highlighted in the updated safety report.

To further complicate matter, the database was subject on both occasions (the original CSR and the 120-day update) to cut-off dates, a practical solution to allow a still-live database to allow phased cleaning of the data to permit interim reporting events. This meant that data after a cut-off date would be ineligible for reporting.

So for example, given a cut-off date of 30JUN2011, if an AE is reported with an start date of 28JUN2011 and an end date of 01JUL2011, then this record in the database would have to be rolled-back to show that the AE was still “Ongoing”.

We were provided with uncut data for both reporting occasions, and so it became necessary not only to identify the observations in the database that changed, but also the reason why the data had changed. These changes in the database may have been due to either data not being entered or amended since the original date, or the application of the two respective data cuts on the two dates.

DESIGN OF THE SOLUTION

The SDTM data produced for the original CSR was generated from the applying a data cut to the raw data, but also uncut versions were produced. For the 120-day safety updates, a similar strategy was proposed. However, with two different data cuts and two different slices of the database, standardising the SDTM was the first task.

Basically, the generic processes identified were

- Generate SDTM for each variation of data slice and data cut
- Generate Supplementary domains if required
- Apply relevant data cut (if required) to parent and supplementary domains
- Compare each variation of generated SDTM data and create flags from the results of that comparison
- Add flags to the previously generated SUPP domain. If it didn’t exist then create one.

The final generated SDTMs (with flags) would then be used to create ADaM data sets for the generation of listings and tables for the safety reports. Listings would include the flags to identify changes in the data.

STEP 1: CONVERT SDTM GENERATION PROGRAMS INTO MACROS

The simplest bit! The original S_XX programs were adapted by wrapping them top and bottom with the macro header and footer. Library references in the code were also changed to the newly created macro parameter variable.

```
%let progname=S_AE;
%logfile(switch=ON);
%m_clear;
* read raw data from RAW.AEC;
proc sql noprint;
  create table AE1 as select
    AEC.STUDYID as STUDYID
  ,"AE" as DOMAIN
  ,compress(AEC.STUDYID)||"-"||put(AEC.SUBJID,z8.) as USUBJID
  ,left(put(AEC.AESPID,best.)) as AESPID
  ,AEC.AESPID as SORTORD
  ,AEC.AETERM as AETERM
```

Figure 12. SDTM program extract prior to conversion

```
%macro makeAE(rawlib = raw
             ,outds = sds.AE
             ,suppds = SUPPAE1
             );

* read raw data from RAW.AEC;
proc sql noprint;
  create table AE1 as select
    AEC.STUDYID as STUDYID
  ,
    "AE" as DOMAIN
  ,
    compress(AEC.STUDYID)||"-"||put(AEC.SUBJID,z8.) as USUBJID
  ,
    left(put(AEC.AESPID,best.)) as AESPID
  ,
    AEC.AESPID as SORTORD
  ,
```

Figure 12. SDTM program converted to a macro

We also needed a new S_xx program that called the newly created macro…

```
%let progname=S_AE;

%logfile(switch=ON);

%m_clear;

%makeSDTM(domain = AE

         ,suppvars = LLT AEOUTC

         ,datevar = AESTDTC AEENDTC

         ,suppdatevar =

         ,dayvar = AESTDY AEENDY

         ,compareID = studyid usubjid visit aebodsys aedecod aespid

         );

%logfile(switch=OFF);
```

Figure 13. New S_xx program that calls the original macro-ised program

STEP 2: CREATE GENERIC MAKE SDTM MACRO

The generic application of the data cut itself was moved outside of the individual SDTM macros, and became an identifiable generic process applied to all data sets. It was decided to identify each variation of the SDTM data by creating many different SDS libraries with a prefixes to tell them apart, with the prefixes identified as comments in the macro code.

The macro calls to each generated SDTM could be run dynamically by a creative use of a macro parameter. For instance, the program code has a call to the macro using the command %make&domain – so when the macro is called with the parameter domain set to AE, then the macro %makeAE is executed.

```
****************************************;
* SDTM Creation                        *;
* (1) ZZ prefix                        *;
*   - 4MSU data with no datacut        *;
****************************************;
%make&domain;
%if &numSUPP %then %do;
  %makeSupp;
%end;
****************************************;
* SDTM Creation                        *;
* (2) no prefixes                      *;
*   - 4MSU data with 30NOV2009 datacut *;
*   - reading from the ZZprefix SDTMs  *;
****************************************;
%datacut_date;
%if &numSUPP and &domain ne AE %then %do;
  %makeSupp;
  %datacut_date;
%end;
****************************************;
* SDTM Creation                        *;
* (3) CC prefix                        *;
*   - 4MSU data with Cycle 1 datacut   *;
*   - reading from the ZZprefix SDTMs  *;
****************************************;
****************************************;
* SDTM Creation                        *;
* (4) MM prefix                        *;
*   - CSR data with 30NOV09 datacut    *;
*   - reading from oldC1raw libname    *;
****************************************;
****************************************;
* Create Observation Flag 1:           *;
* (5) Compare Old data (with C1 cut)   *;
*   with new data (with C1 cut)        *;
*   Identify changed observations $     *;
*   Identify new observations     £     *;
****************************************;
%makeCompare(domain=&domain
            ,inds=oldc1sds.&domain
            ,compds=sds.CC&domain
            ,outds=change&domain.1
            ,idvars=&compareID
            ,flagvar=obsfl1
            );
****************************************;
* Create Observation Flag 2:           *;
* (6) Compare Old data (with 4MSU cut) *;
*   with new data (with 4MSU cut)      *;
*   Identify the changed observations $ *;
*   Identify the new observations     £ *;
****************************************;
%makeCompare;
Etc.
```

Figure 14. New generic MAKESDTM macro

A note here on coding style – it was necessary to create a COPYDS macro – this way data sets could be copied over with all their attributes intact (especially data set label) as this would be crucial for the eventual submission of the data sets. By using a SET statement in a data set this would remove all the previously created attributes.

```
*************************************;
* SDTM Creation                     *;
* (9) Add flags to final SUPP SDTM  *;
*     (if it exists)                *;
*************************************;
proc sql noprint;
  create table &domain.flags as
  select &compSqlList,
         &domain..&idvar,
         pre.obsfl1,
         pre.obsfl2,
         pre.obsfl3,
         pre.obsfl4
  from sds.&domain left join pre_&domain.flags as pre
  on &compSqlOn;
quit;
%makeSupp(domain=&domain, idvar=&idvar, inds=&domain.flags, outds=flagsupp&domain,
qnam=obsfl1 obsfl2 obsfl3 obsfl4);
%if &numSUPP %then %do;
  proc append base=sds.supp&domain data=flagsupp&domain;
  run;
%end;
%else %do;
  %copyDS(inds=flagsupp&domain, outds=sds.supp&domain);
%end;
```

Figure 15 MAKESDTM macro (continued) with COPYDS reference.

STEP 3: COMPARE DATA SETS AND GENERATE FLAGS

Of most relevance to this paper! Following SORTs of the two data sets, PROC COMPARE is invoked with the options OUTBASE, OUTCOMP and OUTDIF, and a NOPRINT with OUT=. This was the first time that I'd used the PROC COMPARE with a NOPRINT option, and had to force myself to learn about the output from the COMPARE procedure.

I wanted to create different flags to distinguish between the new observations and those that had merely changed.

This first data step singled out only those observations WHERE _TYPE_ EQ "DIF". New or redundant observations were isolated in the code by using a WHERE _TYPE_ NE "DIF",

Another comment here too on coding style - a CALL SET was used here to avoid using the technique of counting the variables and using macro code to loop through them. In this way I could do the same thing inside a data step and was more efficient as well.

For the identification of changes to observations, then the DIF record was scrutinized. Whereas in the BASE and COMPARE records these contain the variable values in each data set, the DIF record stores the differences between the two data sets in each variable in that record. Where a difference occurs in a character variable, this is indicated by a X masked with dots. Differences detected between the two data sets for numeric variables are stored in the same numeric variable, shown as the arithmetic difference between the values.

For the identification of new observations, it was a much simpler matter of identifying unique instances of BASE or COMPARE records. When both BASE and COMPARE records exist, there follows a DIF record as well – meaning that both observations exist in the BASE and COMPARE datasets, but with differences. When only a BASE or a COMPARE exist in the output data set, then this indicates that the observation exists only in the BASE or COMPARE data set. For the purposes of identifying new data, the COMPARE record was of more interest - as it would be difficult to add a flag to a record that doesn't exist! To reflect the currency theme of flagging with a $ for differences, I used a £ to show the new observations.

```
proc compare data=inds compare=compds out=compset(drop=_obs_)
  noprint outbase outcomp outdif;
    id &idvars;
run;

data dif;
    &attrib _TYPE_ $8;
    label &flagvar="&flaglab";
    dsid1 = open('compset(where=(_TYPE_ eq "DIF"))');
    call set(dsid1);
    numVars = attrn(dsid1, "NVARS");
    numObs = attrn(dsid1, "NOBS");
    do obsloop = 1 to numObs;
      &flagvar = '';
      rc = fetchobs(dsid1, obsloop);
      do varLoop = 1 to numVars;
       _varname_ = upcase(varname(dsid1,varLoop));
       if _varname_ not in (&IDinList '_TYPE_' "&noComp") then select
vartype(dsid1,varLoop));
         when ('C') if index(getvarc(dsid1, varloop), 'X') then do;
           &flagvar = '$';
         end;
         when ('N') if getvarn(dsid1, varloop) not in (., 0) then do;
           &flagvar = '$';
         end;
         otherwise;
       end;
      end;
      if not missing(&flagvar) then output;
    end;
    keep &idvars &flagvar;
  run;

data newobs oldobs;
  &attrib _TYPE_ $8;
  label &flagvar="&flaglab";
  set compset(where=(_TYPE_ ne "DIF"));
  by &IDvars _TYPE_;
  if first.&&id&numID eq last.&&id&numID then do;
    if _TYPE_ eq "COMPARE" then do;
      &flagvar = "£";
      output newobs;
    end;
    else if _TYPE_ eq "BASE" then output oldobs;
  end;
  keep &idvars &flagvar;
run;
```

Figure 16 Adding the flags

STEP 4: GENERATE ADAMS AND NEW LISTINGS WITH FLAGS

Patient ID/ Age(yr), Sex, Race	Visit	Test Performed	Date of Assessment	Study Day	Laboratory Test	Unit	Result	Chg^
10011006/ 53,M,W	CYCLE 2 DAY 1	Yes	2009-06-02	26	Bacteria		ND	
					Casts		ND	
					Crystals		ND	
					Epithelial cells		ND	
					Glucose	Not Applicable	NEG	
					Ketones		NEG	
					Occult blood		ND	
					pH		6	
					Protein	Grams per 24 hours	+/-	
					RBC	Picogram	+	
					Specific gravity	Kilogram per Liter	1.03	
					WBC	Picogram	+	
	STUDY TERMINATION	Yes	2009-06-23	47	Bacteria		NEG	_£
					Casts		NEG	_£
					Crystals		NEG	_£
					Epithelial cells		NEG	_£

(CONTINUED)

^Changes to Listing from the Study Phase Listing are identified as follows; $_= changed observation in data,
_$= change as a result of removing original Study Phase cut assumptions,
$$= changed observation in data and change as a result of removing original Study Phase cut assumptions,
£_= New Observation (Study Phase), _£ = New Observation (since Study Phase).
Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx/4MSU/PROD/pg/Listings/l_uri1.sas FINAL 25MAR2010:18:21

Figure 16 The final listing generated from the ADaM

CONCLUSION

PROC COMPARE is very powerful and I have only scratched the surface in what it can do. The SAS Procedures guide is a great source of reference for many other options available to you, and goes into greater detail than I have been able to do here. The practical application of using the COMPARE output data sets was a great learning curve and also a technique to solve a problem that arises frequently in the pharmaceutical industry. I hope that you will all now find time to GO COMPARE!

CONTACT INFORMATION <HEADING 1>

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

Michael Auld
Ampersoft Ltd
London
E-mail: sas@w3mac.com