Paper CS05

SAS – How to standardize solutions to recurrent issues

Tangi Sanséau, ICON Clinical Research, Dublin, Ireland
Giuseppe Di Monaco, ICON Clinical Research, Dublin, Ireland

ABSTRACT

As the Pharmaceutical Industry looks at how it can shorten the time from a compound's discovery to its submission to the regulatory authorities, there is an increasing focus on the time it takes to process data and provide programs for statistical analysis. Creating re-usable programs is one of many approaches but this might not be as easy as it looks as database structures and statistical requirements can vary enormously from study to study, compound to compound and therapeutic area to therapeutic area.
Moreover, this variation takes on an additional level of complexity for CROs who also have to think in terms of code that can work from Sponsor to Sponsor.

Whilst Data Standards are one element in solving this issue, the other key component remains in developing flexible macro code which receives parameters from external to the program itself, to deal with the variations in both input and output. Our ultimate goal is to make code reuse simpler and therefore productivity higher regardless of the type of study we are asked to deliver against.

This paper investigates some of the techniques that we have used, within ICON, and provides examples of different approaches that we have taken to pass information to and from SAS macro programs. We aim to demonstrate how different tables or datasets can be produced, or different data structures handled.
Instead of programming macros on a compound basis, we expanded the capabilities of macro programs with the use of external information (compound-specific) which ensures re-usability and standardization across projects.
The examples presented in this paper deal with handling of missing data points within datasets transfer, handling of treatment groups and their population counts within Tables programming and finally re-usability of standard listing/tables programming across compounds/sponsors (i.e. Adverse Events, Concomitant Medications…).

INTRODUCTION

In our industry, in which a number of processes are repeated, there is clearly a potential to re-use some of the programs that have been previously created. The objective of this paper/presentation is to demonstrate that there are easy ways to standardize code created to solve recurrent issues. Moreover, with the whole industry leaning towards more and more standards (see the CDISC emergence), in ICON, we believe that this type of programs will become more and more robust and improve our competitive advantage. In order to find standard solutions, we have segmented problems into sub-problems. This approach is known as modularity.

RECURRENT ISSUE I – DELETING/STORING BLANK ROWS

Sponsors very often ask for blank rows to be deleted from the transfer of final datasets. When this happens programmers have to write/amend and validate the code for each dataset that has to be sent to the Sponsor. DM has to QC the changes applied to the code. So, why not to have a program which can be applied to any study and for any dataset without modifying any transfer program and saving time and resources?

This example shows a standard solution for deleting/storing blank rows in any dataset and for any study without modifying the code.

In order to find a standard solution to this problem it is very important to understand the concept of a blank row.

<u>Defining a blank row.</u>
In most cases an observation of a dataset is composed by DB enterable variables, DB System variables and other information.

In general we can define a blank row as a row where the content of all database enterable variables are blank as shows the first row in the below table, where B, E and G are DB enterable variables whereas A, C, D and F can be DB non enterable variables or other information.

A	**B**	C	D	**E**	F	**G**
XXX		XXX	XXX		XXX	
XXX	XXX	XXX	XXX	XXX	XXX	XXX

Note: It is a responsibility of the SL together with the Sponsor to give an exact definition of a blank row.

STEP I - SEGMENTING ISSUE INTO SUB-PROBLEM

In order to understand the efficiency of this solution to this problem it is very important to abstract the definition of a blank row and at the same time identifying sub problems. This is necessary if we want to build a standard program.

The following information is study dependant.

- Sub problem 1: Number of datasets
- Sub problem 2: Datasets names
- Sub problem 3: Variable names
- Sub problem 4: Number and names of variable that participate in a blank row

We are going to show the above information for study A_001 and A_002.

Sub problem 1:
For study A_001 we can have X datasets whereas for study A_002 we can have Y datasets, where X and Y is the number of datasets.

Sub problem 2:
Dataset names are different within a study and are also different across studies.

See below an example of the content of SAS libraries for 2 different studies:

Content of Study A_001 folder
- ae
- cm
- comm.
- dm
- eg
- fu
- lb
- mh
- vs
- xray

Content of Study A_002 folder
- Adveve
- Conmed
- Demog
- Ecg
- Lab
- Vital

This illustrates sub-problems 1 and 2 as the number of datasets and their names are different from one study to another.

Sub problem 3:
Variable names are different within a study and are also different across studies.

Sub problem 4:
Number and names of variable that participate in a blank row are dataset-dependant and study-dependant.

See below images which show the content of a “similar” dataset for 2 different studies (which could be named “Ae” and “Adveve” as per previous example)

Study A_001

	SUBJID	VISIT	VISITNUM	SUBEVENT	REPEATSN	LOGINTS	ACCESSTS	LSTCHGTS	
1	0080074	SUMMARY	800	0	1	17MAR2009:05:49:39	17MAR2009:10:33:10	29MAY2009:11:35:29	17M
2	0140026	SUMMARY	800	0	1	05NOV2008:07:05:03	05NOV2008:08:08:18	05NOV2008:08:08:18	05N
3	0140026	SUMMARY	800	1	1	12DEC2008:04:49:11	12DEC2008:05:55:18	12DEC2008:05:55:18	12D
4	0140062	SUMMARY	800	0	1	20FEB2009:10:46:58	23FEB2009:04:57:30	23FEB2009:04:57:30	20F

Study A_002

File Edit View Tools Data Solutions Window Help

	SITE	SUBJNO	SUBJID	AESEQ	AETYPE	AEVT	AESDTD	AESDTM	AESDTY	AESDT	AEEDTD	AEEDTM	AEEDTY	AEEDT	AEONGO	AEFREQ
1	0006	001	0006001		Pretreatment Event	URINARY TRACT INFECTION	18	MAR	2009	18MAR2009	28	MAR	2009	28MAR2009		Continuous
2	0006	001	0006001	0	Pretreatment Event	EXIT SITE INFECTION (PD CATHETER)	31	MAR	2009	31MAR2009	14	APR	2009	14APR2009		Continuous
3	0009	001	0009001		Adverse Event	UPSET STOMACH WITH LIGHTHEADEDNESS	01	FEB	2009	01FEB2009	02	FEB	2009	02FEB2009		Intermittent

STEP II – SOLVING SUB-PROBLEMS/PROBLEM

A solution to these sub problems is to have an external excel file that can be read by the program as shows the below image. The below file will be loaded in to SAS and it be will create a dataset for each sheet with all variables and just one row per dataset (which represents a blank row).

Microsoft Excel - spec.XLS

File Edit View Insert Format Tools Data Window Help — Type a question for help

Reply with Changes... End Review...

A2

	A	B	C	D	E	F	G	H
1	PNO	SCREE_NO	PATNO	CNO	EVENT_ID	PAG_NAME	LAB_DT	LABRF_NO
2							null	null
3								

LAB_CRFB | LAB_CRFC | INEX_CRI | LAB_CRF | LAB_CRFI

Ready NUM

(Note: The above spreadsheet is study dependant and it is created by a program which will not be described in this paper.)

We have decided to assign to the DB enterable variables (that participate in a blank row) a value “null”. Please note that the “null” value has been chosen for describing a blank variable but any other word can be used and of course the program should be amended accordingly.

STEP III – STANDARDIZING THE SOLUTION

This step will show parts of the program and will demonstrate that the code of this program does not require any changes when applied on different studies.

Description of the code.

1) The program gets a list of the dataset names from the library of the delivery datasets. Then it counts the number of dataset in the list and afterward it creates macro variables to store the names of the datasets.

```
*Get list of datasets;
proc sql;
  create table dataset_list as
  select distinct memname
  from cont;
  select count(*) into :dset_count from dataset_list;
quit;


*counting the number datasets;
data count_dataset;
  set dataset_list end=end_dset;
  if end_dset then call symput('n_obs',_n_);
run;

*creating macro variables for each dataset;
```

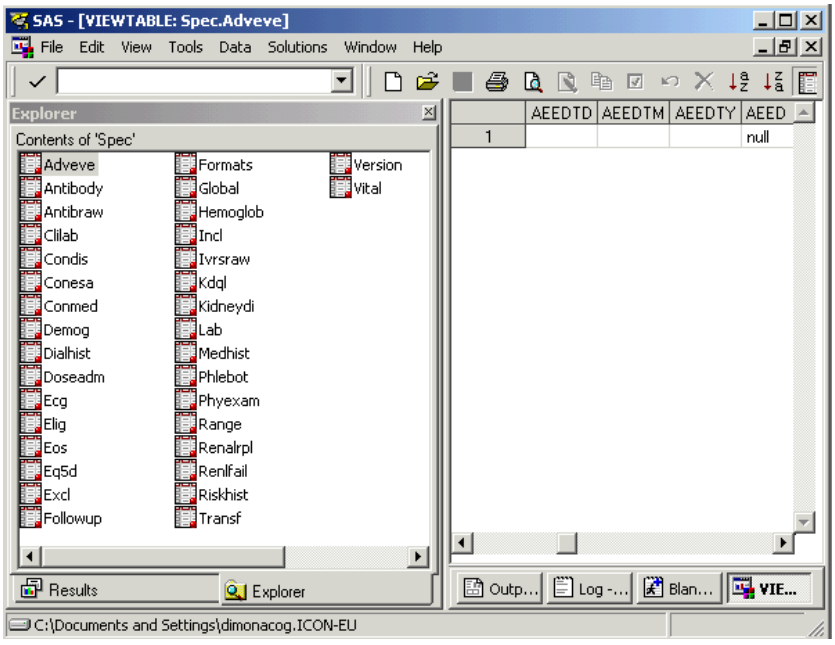
```
data _null_;
  set count_dataset;
  %do k=1 %to &n_obs;
    if &k=_n_ then call symput(compress("MEMNAME"||&k), MEMNAME);
  %end;
run;
```

For example for study Study A_001 the program creates a list of dataset as shown in the below image where n_obs holds the number of datasets and MEMNAME the dataset name.

	Library Member Name
1	ADVEVE
2	ANTIBODY
3	ANTIBRAW
4	CLILAB
5	CONDIS
6	CONESA
7	CONMED
8	DEMOG
9	DIALHIST
10	DOSEADM
11	ECG
12	ELIG

2) At this point the external spreadsheet can be loaded in to SAS.

```
*loading the spec;
   data _null_;
     set count_dataset;
     %do j=1 %to &n_obs;
    %LET filename = &STUDY_FOLDER\Oc\Data\&STATE\CHECKS\BLANK ROWS\DATA SPEC\blank
rows.xls;
       %LET SHEET = &&MEMNAME&j;
       %INC "&STUDY_FOLDER\Oc\Programs\load_excel.sas";
   *moving the datasets to spec library;
       proc copy in=work out=spec move;
         select &&MEMNAME&j;
    run;
  %end;
run;
```



3) The program will now check:
   - The number of dataset loaded from the spec is equal to the number of dataset of the delivery datasets.
   - The dataset names loaded from the spec are equal to the number of dataset of the delivery datasets.
   - Which variables in which datasets (in the spec library) have value equal to “null”.

```
*counting the number of variables for each dataset;
data count;
    set count end=end_dset;
    if end_dset then call symput('n_obs',_n_);
run;

*Creating macro variables for each dataset;
data _null_;
  set Count;
  %do b=1 %to &n_obs;
    if &b=_n_ then call symput(compress("VAR_NAME"||&b), NAME);
  %end;
run;

*Creating macro variables which are null for each dataset;
data _null_;
  set spec.&dset end=end_dset;
  count=0;
  %do l=1 %to &n_obs;
    if &&VAR_NAME&l='null' then do;
      count=count+1;
      call symput(compress('key'||count), compress("&&VAR_NAME&l"));
    end;
  %end;
*Counting the key variables that are participating in the blank rows;
  if end_dset then call symput('COUNT_KEYS', compress(count));
run;
```

4) We now have all info we needed for comparing the dataset created from the spec against the delivery dataset. After the comparison if a blank row is found it gets deleted from the sponsor dataset and the same row is stored in a newly created dataset called blank_<dataset name> to keep track of it.

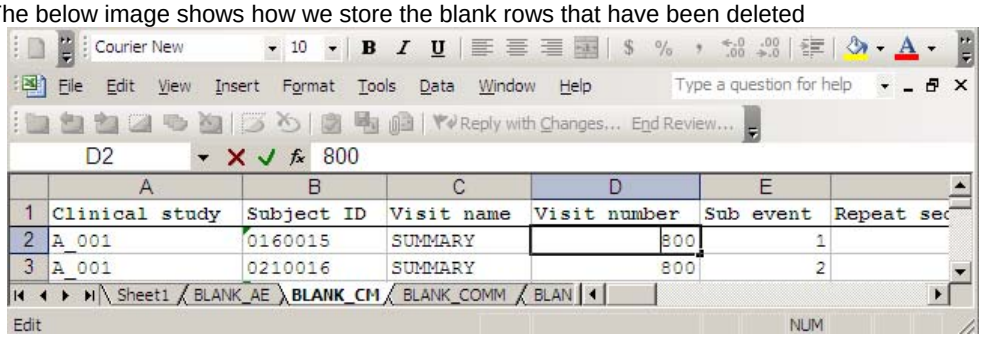
```
*creating the dataset of blank rows to keep track of what is deleted from the sponsor
dataset;
data output.blank_&dset;
    set libref.&dset;
if %do z=1 %to %eval(&COUNT_KEYS-1);
       missing(&&key&z) and
   %end;
       missing(&&key&COUNT_KEYS);
run;

*Deleting the blank rows from the sponsor dataset;
data libref.&dset ;
    set libref.&dset;
    if %do a=1 %to %eval(&COUNT_KEYS-1);
       missing(&&key&a) and
       %end;
       missing(&&key&COUNT_KEYS) then delete;
run;
```

Note: One thing is to be considered. The “null” value in the spec dataset corresponds to blank variable in the delivery dataset.

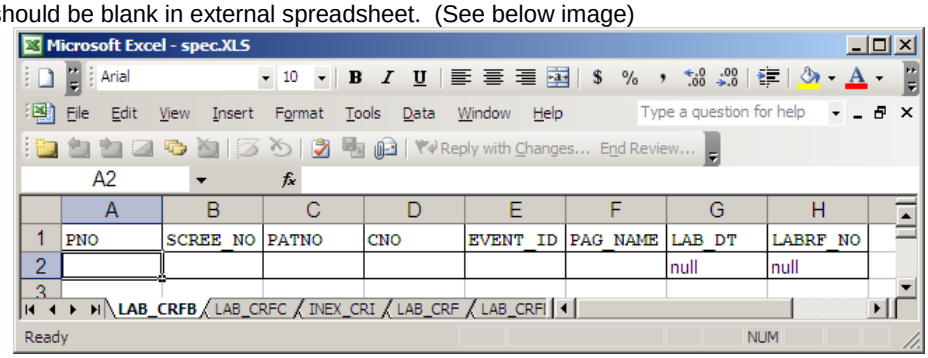
5) Once all datasets have been processed, all newly created datasets called blank_<dataset name> will be saved in to an excel spreadsheet using the same technique we used for loading.

The below image shows how we store the blank rows that have been deleted



	A	B	C	D	E	
1	Clinical study	Subject ID	Visit name	Visit number	Sub event	Repeat sec
2	A_001	0160015	SUMMARY	800	1	
3	A_001	0210016	SUMMARY	800	2	

We would like to emphasize that the only manual process is for the SL to define a blank row and afterward to fill in the variables which should be blank in external spreadsheet. (See below image)



	A	B	C	D	E	F	G	H
1	PNO	SCREE_NO	PATNO	CNO	EVENT_ID	PAG_NAME	LAB_DT	LABRF_NO
2							null	null
3								

RECURRENT ISSUE II – COMPARING DATA TRANSFERS

One of the main tasks for any SAS Programmer and/or biostatistician is often to detect and handle data issues contained in the datasets you are working with for your Listings and Tables production.

While programming our listings and Tables, we might notice from time to time “outliers”, aberrant values in the data which were not spotted by the Data Managers prior to the transfer and should probably be either corrected or at least checked.

When we report these values to Data Management, in case one of them is subject to a change in one of the datasets, the process is that all datasets are re-transferred. In that case, even though you expect only one of the dataset to have changed, you can never ensure that a dataset has been impacted or not without comparing it to the version included in the previous transfer.

Therefore, we thought creating a tool that compares dataset transfers would ideally save time for every programmer and give him the peace of mind to know that checks performed on the previous transfer do not need to be re-done on the new one, since data is similar.

Starting with this idea, you come to the point where you realize that from one study to another, the specificities of a dataset transfer (SDTM or not) can enormously vary. Again, using the same methodology as in the two previous examples, we will describe how we created a standard program that works properly for any client, any compound, any study, thanks to the external use of the study-specific information.

STEP I - SEGMENTING ISSUE INTO SUB-PROBLEM

Here is a list of items that we thought could differ from one transfer to another and that we should therefore handle one after the other as “sub-problems”, prior to bringing the solutions all together as a “standard” solution:

- Item 1: number of datasets: obviously, if a dataset was included in only one of the 2 transfers to be compared, no comparison can be done.
- Item 2: name and structure of the datasets: the example described here does not take into account the possibility that a dataset can change name from one transfer to the other. We assume that the name of a given dataset has been setup and does not change in the course of a project. However, such a feature could be implemented in our solution. Similarly, the additional step that is not included in our program would be to consider the complete Specification of the transfer as the external source, hence, reading the name and structure of all dataset, in order to determine more accurately why would a dataset be present in only one of the transfer or why is there a variable in dataset A that was not part of the same dataset previously

- Item 3: name of the variables (and key identifiers: finally, if we know that a dataset is in the new transfer was also included in the previous one **and** it is also confirmed that the structure of the dataset is similar in both transfers, we can then focus on the data itself and compare

STEP II – SOLVING SUB-PROBLEMS

As stated in the split of the main issue into sub-problems, there are many ways of solving the sub-problem related to the name and structure of the dataset, the most appropriate one would probably being to read the Dataset Transfer Specification and import into SAS that information. That could also be a basis to solve sub-issue 1 "number of datasets" since we know what to expect from a transfer. If using that solution, we can note that we could even check the structure of the dataset (meaning the number of variables, name of the variables, length, format, informat, label…).
However, we decided to focus on the data itself and therefore, for our example, we'll only base our program on the list of datasets and their corresponding key variables.
The list is the external information that is read by the macro program and that allows the proc compare to be efficient.

STEP III – STANDARDIZING THE SOLUTION

As stated earlier, an ideal solution would be to enlarge that vision by directly reading the datasets specifications, if such a document exists.
Doing so, one of the first issues that would come in mind would be the format of the document. Some datasets Specifications documents can be in Word, Excel, or xml format depending on the company, the study, the requirements…
Using our method, this issue would be the first of the sub-problem that we could resolve by converting any document into SAS. However, when standardizing the solution, we would probably then think that an additional step to convert the word document or the Excel document into xml format could be added so that all Specs were then handled the same standard way from that point.
This example shows again the power of the method.

RECURRENT ISSUE III – "MACRO-IZING YOUR CODE" WITHIN CLIENT/COMPOUND SPECIFIC STUDIES

In this example, we will see that some of the code used in the creation of Listings, Tables or Figures can be centralized in external macros instead of being repeated at the top of every program.
For instance, all Tables need the N to be displayed in the header of the column, therefore all Table programs will contain a step in which those N will be calculated.
The solution is to define this recurrent code as a macro (in this case, named "N_Pop"):

STEP I - SEGMENTING ISSUE INTO SUB-PROBLEM

This can be split into the following sub-problems:
- No. 1: How many treatments in the study (as this varies from study to study)?
- No. 2: Will some tables need to display a "Total" column gathering/summarizing all patients?
- No. 3: How many different populations will be used in the creation of these Tables?

STEP II – SOLVING SUB-PROBLEMS/PROBLEM

For a compound that is tested against placebo (therefore 2 different treatment groups), we could use the following macro, where Treatment 1 is the Placebo, Treatment 2 is the active product:

```
*Using the name of the treatment variable as the parameter;
   %macro N_Pop(trt=);
   %global ntrt1 ntrt2;

*Creating macro variables to contain treatment information;
   data trt;
          set libnew.d_master;
          trt       = &trt;
   run;

   proc sort data=trt;
          by trt subjid ;
   run;
```

```
*Creating count per treatment;
   data _null_;
     set trt;
     call symput('ntrt'||compress(put(trt, best.)),
       compress(put(count, best.)));
   run;
   %mend;
Note: This is considering adsl as the “key” analysis dataset as per current ADaM
Standards;
```

Regarding sub-problem No. 2, we can decide to create a “Total” treatment group independent on the fact that we would need it or not, but in order to be able not to use it, we would also create a macro variable containing the number of treatment groups including a total category. This enables us not to take into account the total (for instance by defining as a standard that the last numerical treatment group is the total).

In order to fix the sub-problem 3, the use of different population depending on the listing/table we need to produce, we can simply add the population flag as a parameter in the macro call.

STEP III – STANDARDIZING THE SOLUTION

From the above example, if we take into account all subproblems, we could even include more functionalities such as the creation of standard column header as it is often consistent across all outputs of a study. The objective is not to develop such fonctionalities, bu the example below is based on the fact that the character treatment variable has the same name as the numeric treartment variable with a “c” as a suffix (as per current ADaM standards).

The final macro can therefore look like the following “N_pop_total”:

```
   %macro N_pop_total(trt=, pop=);

   %global tot;

   *Defining the name of the character variable based on the name of the numeric variable;
   %let treatment=&trt.c;

   *Creating macro variables to contain treatment information;
   data trt;
      set libnew.adsl(where=(not missing (&trt) and &pop=1));
      treatment = &treatment;
     trt       = &trt;
     output;
     trt=&tot;
     treatment=’Total’;
      output; *Outputing all observations a second time as “Total”;
   run;

   proc sort data=trt;
          by trt subjid ;
   run;

   *Looking for total number of treatments;
   data _null_;
       set trt;
       call symput('tot',compress(put(trt+1,best.)));
   run;

   *Defining ntrt&i as containing the N for each treatment group (including Total) and
   labeltrt&i as the corresponding label;
   %do i=1 %to &tot;
          %global ntrt&i labeltrt&i;
   %end;
```

```
*Looking for total number of subjects per treatment;
*Calculating frequency for treatment;
proc freq data=total noprint;
    tables trt*treatment/ out=Totalfreq;
run;

*Creating label per treatment;
data _null_;
  set Totalfreq;
  call symput('labeltrt'||compress(put(trt,best.)),
    left(trim(treatment))||'¬(N='||compress(put(count, best.))||')');
  call symput('ntrt'||compress(put(trt,best.)),
    compress(put(count,best.)));
run;

%mend;

Note: This macro considers adsl as the "key" analysis dataset as per current ADaM
Standards;
```

From this example, the use of labeltrt&i variables will create the column headers as follows:

	Placebo (N= 100)	**Active Product (N= 99)**	**Total (N= 199)**
Mean	X.X	X.X	X.X
Std Error	X.XX	X.XX	X.XX

RECURRENT ISSUE IV – RE-USING TABLES/LISTINGS/FIGURES PROGRAMS ACCROSS CLIENTS/COMPOUNDS

STEP I - SEGMENTING ISSUE INTO SUB-PROBLEM

The objective of any cost-saving solution is to ensure that the SAS code created for one clinical study can be re-used for others wherever possible. This second example will focus on how to re-use code created for one compound (or one project) across other compounds.

The key for SAS code to be re-usable are the following:
- ensure external macros are "generic", i.e. as standard as possible, and as less "compound-specific" as possible
- try to minimize the occurrence of the compound specific information

Here are a few examples:
For the production of Adverse Events Tables for the Study A_001, you need to use the randomized treatment. However, for the production of Adverse Events Tables for another study B_001 that you are working on in parallel, you need the actual treatment to be used.

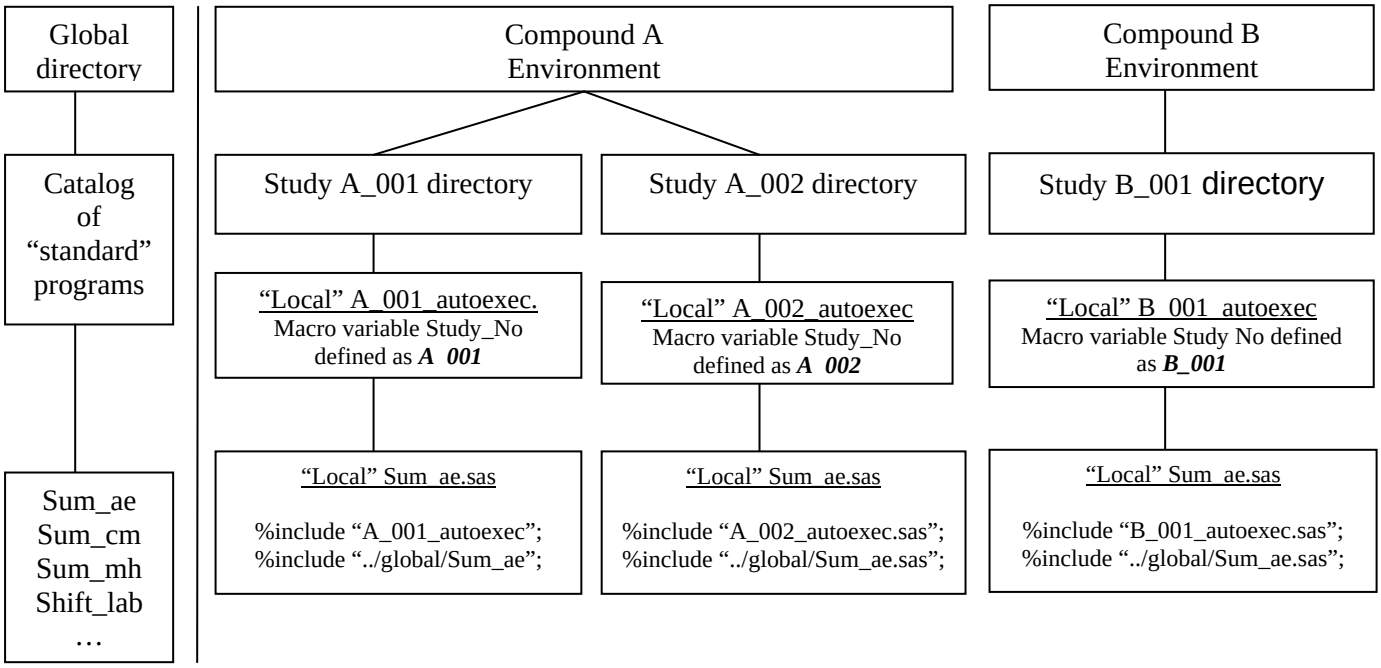
This can be done by an external file/macro.

STEP II – SOLVING SUB-PROBLEMS/PROBLEM

It is common in the industry to have external "setup" files such as libassign, autoexec… These files can be made more efficient by including as much information as possible i.e. study number, client study number, treatment to be used for safety tables, efficacy tables (can be defined as treat_safety= trt1randomized, treat_efficacy= trt1actual, treat_listing= trt1randomized etc.

This way, at the top of each of your table programs, you define for once and only once in the program, the treatment variable to be used, i.e. %let treatment_variable= &treat_efficacy for efficacy tables or %let treatment_variable= &treat_listings. The global location is then responsible for assigning those treatment variables on a study-specific basis.

Here is an example on how to create a Table summarizing the Adverse Events, for various studies using a standard catalog of macros previously developed, represented by the following diagram:



In the above example, "standard" programs are located in global folder.

STEP III – STANDARDIZING THE SOLUTION

Here is an example of what can be included in sum_ae for the A/001 study:

```
%include "A_001_autoexec.sas";
```

This include statement executes the autoexec specific to the A/001 study, hence setting up your SAS environment to work in the appropriate 001/001 folders and setup a few macro variables as A_001 that can then be used in the "generic" programs.

```
%include "../global/Sum_ae.sas";
```

The second include statement then executes the generic macro to create the Table. Thanks to the macro variables that were initialized and setup in the study specific autoexec, some of the code from sum_ae standard program can still be made study specific by using statement such as:

```
%if &study_no = A_001 then do;
      %let ntrt = 6;
      %let sort_order = soc_term descending c&ntrt;
%end;
%if &study_no = B_001 then do;
      %let ntrt = 3;
      %let sort_order = soc_term pt_term;
%end;
```

In this example, the output for study A/001 will be sorted alphabetically by SOC term and by decreasing frequency of Preferred Term within that SOC category, whereas for study B_001, the Preferred Term will be displayed alphabetically within the SOC category.
Also, we were able to define the number of treatment (hence number of columns to be displayed in the output) on a study basis. This allows us to use those specificities in our proc report, as follows:

```
proc sort data=ae out=final; by &sort_order; run;

data final_output;
    set final;
    ord=_n_;
run;
```

```
/*Assuming we have 60 characters to display all treatment columns*/
%if &ntrt=2 %then %do; %let width_col=30; %end;
%if &ntrt=3 %then %do; %let width_col=20; %end;
%if &ntrt=4 %then %do; %let width_col=15; %end;
%if &ntrt=5 %then %do; %let width_col=12; %end;

proc report data=final center;
  column ("__" " " ord soc_term pt_term (c1-c&ntrt));
  define ord     /order order=internal noprint;
  define soc_term/display width=25 "SOC Term";
  define pt_term /display width=25 left flow "Preferred Term";
  %do i=1 %to &ntrt;
    define c&i /display width=&width_col center "&&label&i";
  %end;

    break after ord/skip;
run;
```

CONCLUSION

The objective was to propose a few examples of coding standard solutions.
We would like to emphasize that creating these solutions is a long-term investment. Indeed, creating standard solutions is more time-consuming that creating study or compound-specific programs in the first place but the re-usability and robustness of these standard solutions demonstrate to be cost-saving in the long term.

CONTACT INFORMATION

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

Giuseppe Di Monaco
ICON Clinical Research
South County Business Park
Leopardstown
Dublin 18
Ireland

Work Phone: +353 1 291 2296
Fax: +353 1 291 2723
Email: Giuseppe.DiMonaco@iconplc.com
Web: www.iconplc.com

Tangi Sanséau
ICON Clinical Research
South County Business Park
Leopardstown
Dublin 18
Ireland

Work Phone: +353 1 291 2483
Fax: +353 1 291 2723
Email: Tangi.Sanseau@iconplc.com
Web: www.iconplc.com