



Code Evolution

How Programs are Developed and Refined Over Time

Pantaleo Nacci, Head Statistical Reporting

Berlin, 19 October 2010

Agenda

Introduction

Overall Organization

The Final Goal: Standard Structures

Study Dimension

Data Domain Dimension

How I Did It

Moving from SASHELP to PROC SQL

The Power of CALL EXECUTE

Why You Should Use Macro Language

Writing Forward-looking Code

Conclusions

Introduction

- Finding new patients/subjects for clinical trials is increasingly difficult, as well as expensive
- Many companies have in-house data from studies going back many years, even if usually collected using different CRF and data standards
- During the recent A/H1N1 pandemic, NVD received from Regulatory Agencies many requests for retrospective safety analyses of all data collected in selected trials, some dating back to 1993
- Most answers were obtained using a data mart I created over the last 6 years, currently containing data from more than 130 influenza studies

Agenda

Introduction

Overall Organization

The Final Goal: Standard Structures

Study Dimension

Data Domain Dimension

How I Did It

Moving from SASHELP to PROC SQL

The Power of CALL EXECUTE

Why You Should Use Macro Language

Writing Forward-looking Code

Conclusions

Overall Organization

- Pooling of studies had already been done before I joined the company, but the approach used was an ,on-the-fly‘ one, so that it might also be impossible to recreate the same outputs at a later stage
- On top of that, in several cases common code had not been updated everywhere
- Looking back to my previous experiences with data pooling, I then decided early on to
 - create static copies of the pooled data to allow reproducibility
 - use a matrix approach for the programs, to maintain them lean and easy to maintain

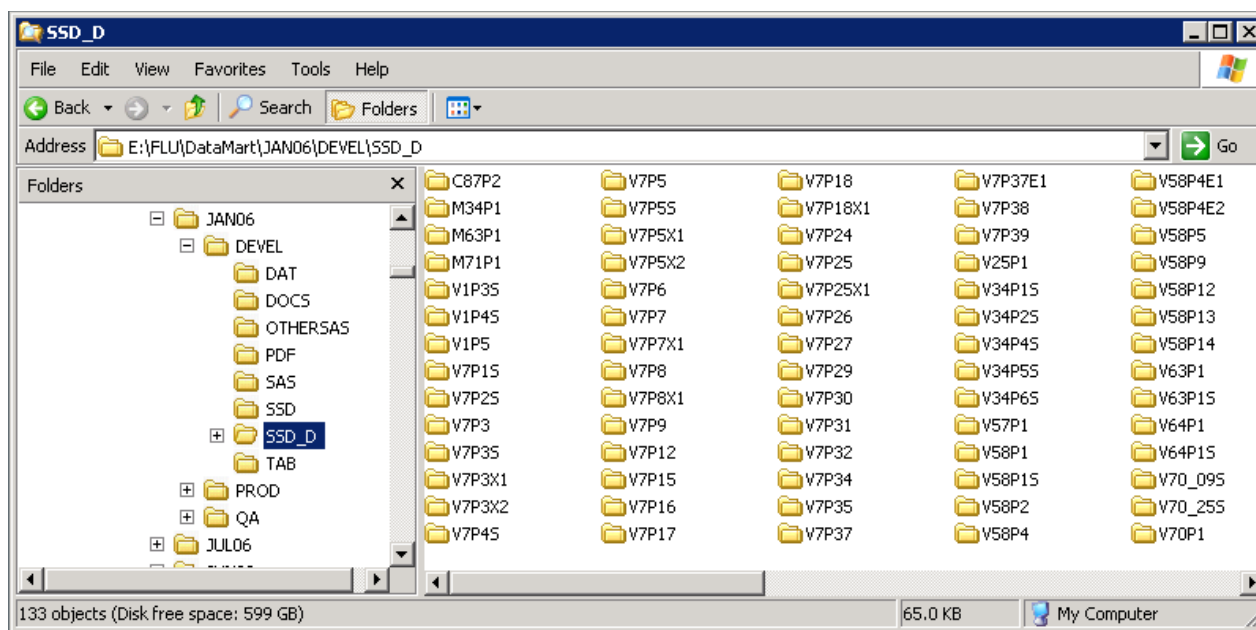
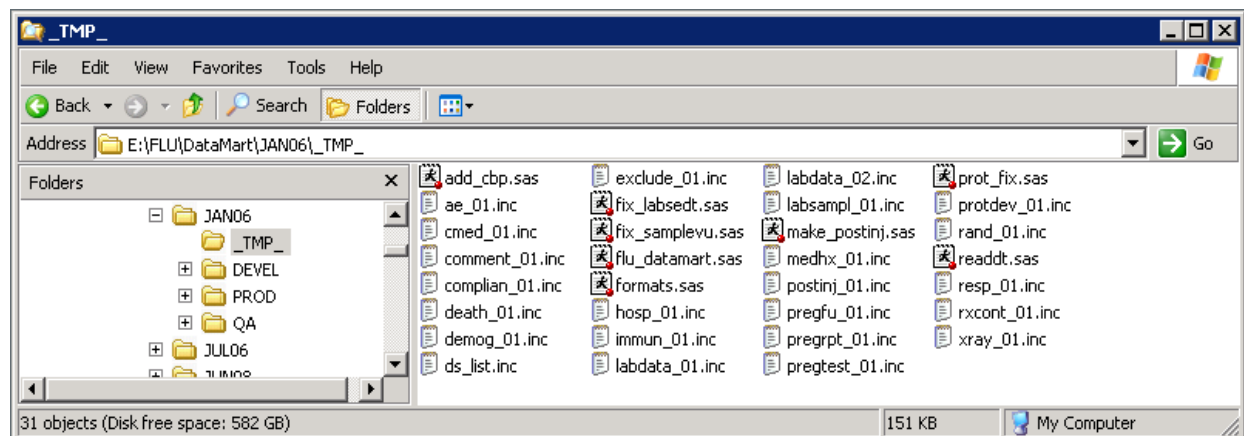
Overall Organization

The final goal: standard structures

- Since a good internal standard was already in use and this was after all, at least initially, just my ,pet‘ SAS project, I did not take into consideration other options, like CDISC
- The Chiron/Novartis standard I had to deal with was designed in the early '90s, and clearly a lot of thought had been put into it since it has remained basically unaltered
- Some global changes had been applied over time, e.g., all variables containing the month part of a date (with a suffix 'MO') had been changed from numeric to character
- Last but not least, I created a directory structure which would allow further expansion, in terms of both studies and data domains covered

Overall Organization

Files and directories



Overall Organization

Study dimension

- Study-specific variations to the current standard existed, in terms of both variables and data domains, and initially they were all dealt with within the study-specific programs
- For the most common manipulations I created a central %SETUP macro, which grew over time from 40 lines to the current 218, as I started identifying repeating patterns and ,families' of studies, thus moving more and more code out of the study-specific programs and into it
- Access to the original CRFs was fundamental to identify which information was collected how in the various studies and avoid misinterpretations

Overall Organization

Study dimension: typical structure of a study-specific program

```
%LET etude = v999_99;

LIBNAME ssd "!&project\&etude.\FINAL\PROD\SSD\" ACCESS=readonly;

* Study-level temporary formats ;
PROC FORMAT;
VALUE $perno
'30M' = 1
...
;
RUN;

%setup;

Study-level changes ;
%prot_fix(ds_in = comments,
          prefix = cmt);
%add_cbp;
...

%LET select=%STR(
WHEN ('A') tgroup = '99';
);
%LET selectr=%STR(
WHEN ('A') rtgroup = '99';
);
%INC 'rand_01.inc';
```

Overall Organization

Data domain dimension

- Since I didn't know how much variability I would find in the studies, I devised a simple filename convention allowing for several versions of the program dealing with same-named data sets
- Until now, that was only needed when dealing with safety laboratory data
- The initial version of the application dealt with ten data domains (adverse events, demography, medical history, concomitant medications, etc.) and it is now up to 20
- Not all data ever collected are currently dealt with, but expansion would be relatively straightforward

Overall Organization

Data domain dimension: typical structure of a domain-specific program

```
%MACRO ds_exist;
%LET dsid = %SYSFUNC(OPEN(death, is));
%IF &dsid %THEN %DO;
%LET rc = %SYSFUNC(CLOSE(&dsid));
DATA death;
    MERGE death (IN = a) out.random (KEEP = prot ext center ptno tgroup);
    BY prot ext center ptno;
    IF a;
RUN;

DATA death (LABEL = 'Death report data' KEEP = prot ext center ptno tgroup ...);
    LENGTH prot $ 18 ...;
    SET death;
    IF COMPRESS(deathdt_) = '---' THEN deathdt_ = '';
    ATTRIB
        prot                                LABEL='Protocol code'
        ...;
RUN;

PROC SORT DATA = death OUT = out.death;
    BY prot ext center ptno;
RUN;

PROC datasets LIB = work MT = data;
    DELETE death;
RUN; QUIT;
%END;
%MEND ds_exist;
%ds_exist;
```

Agenda

Introduction

Overall Organization

The Final Goal: Standard Structures

Study Dimension

Data Domain Dimension

How I Did It

Moving from SASHELP to PROC SQL

The Power of CALL EXECUTE

Why You Should Use Macro Language

Writing Forward-looking Code

Conclusions



How I Did It

- Gaining access to old CRFs (documentation in general) was a major hurdle, but there is no real alternative
- A list of all studies to be taken into consideration is helpful
- The main logical flow is actually quite simple:
 - Define the list of all studies to be included in the data mart (&LIST)
 - Loop through &LIST (or a subset, &LIST_PART)
 - Include the program specific to the study being standardized
 - Include a file to scan through and create the known data domains
 - Merge randomization info, attach labels and formats, and create permanent data sets in the study-specific directory
 - Loop through &LIST to pool the now-standardized data sets
 - Recode AEs, medications, etc. using a common dictionary

How I Did It

Excerpt from DS_LIST.INC

```
%* Demographic and baseline data *;  
%inc'demog_01.inc';  
  
%* Medical history data *;  
%inc'medhx_01.inc';  
  
%* Lab samples collection data *;  
%inc'labsampl_01.inc';  
  
%* Vaccine administration data *;  
%inc'immun_01.inc';  
  
%* Local & systemic reactions data *;  
%inc'postinj_01.inc';  
%inc'rxcont_01.inc';  
  
%* Adverse events data *;  
%inc'ae_01.inc';  
  
%* Hospitalization data *;  
%inc'hosp_01.inc';  
  
%* Death report data *;  
%inc'death_01.inc';  
  
%* Concomitant medications data *;  
%inc'cmed_01.inc';  
...
```

How I Did It

Moving from SASHELP to PROC SQL

- The first problem to solve was how to identify which data sets had been created for the single studies, and more
- Initially I used SASHELP, containing all kinds of info automatically maintained by the current SAS session
- Accessing the VTABLE and VCOLUMN data sets I was able to create a list of existing data sets and check their contents (e.g., variable names and types)
- As the number of studies increased, the time needed to access SASHELP became too long, so I needed an idea
- Moving to PROC SQL maintained the same logic, but with an incredible gain in speed, from minutes to seconds!

How I Did It

Moving from SASHELP to PROC SQL: contents of SASHELP

Explorer

Contents of 'Sashelp'

Table Templt Treport Vallopt Vcatalog Vcformat Vchkcon

Vncolu Vcntabu Vcolumn Vdctnry Vengine Verbmgr Vextfl

Vformat Vgopt Video Vidmsg Vindex Vlibnam Vmacro

Vmember Voption Vrefcon Vrememb Vsacces Vscatlg Vslib

Vstable Vstabvw Vstyle Vsview Vtabcon Vtable Vtitle

Vview Web Webadmn Webaps Webeis Webmsg Webprog

VIEWTABLE: Sashelp.Vextfl

	Fileref	Path Name	Eng
1	#LN00005	TERMINAL	
2	#LN00007	C:\Documents and Settings\naccipa1\My Documents	
3	#LN00009	C:\Documents and Settings\naccipa1\Desktop	
4	MACRO	E:\BCDM\SW_SYS\MACRO\PROD\SAS	
5	MAC_DEV	E:\BCDM\SW_SYS\MACRO\DEVEL\SAS	
6	CDM	E:\BCDM\CDM\MACRO\PROD\CODE	
7	DAT	E:\FLU\DataMart\JAN06\DEVEL\DAT	
8	SAS	E:\FLU\DataMart\JAN06\DEVEL\SAS	
9	TAB	E:\FLU\DataMart\JAN06\DEVEL\TAB	
10	PDF	E:\FLU\DataMart\JAN06\DEVEL\PDF	
11	_DIR_	chdir	
12	#LN00001	C:\Program Files\SAS\SAS 9.1\core\sasmsg	
13	#LN00001	C:\Program Files\SAS\SAS 9.1\access\sasmsg	
14	#LN00001	C:\Program Files\SAS\SAS 9.1\graph\sasmsg	
15	#LN00001	C:\Program Files\SAS\SAS 9.1\intnet\sasmsg	
16	#LN00001	C:\Program Files\SAS\SAS 9.1\sta\sasmsg	
17	#LN00002	TERMINAL	
18	#LN00003	C:\Program Files\SAS\SAS 9.1\connect\saslink	

Editor - Untitled2 *

```
filename _dir_ pipe 'chdir';
```

Output - (Unti... Log - (Untitled) Editor - Untit... VIEWTABLE: S...

Autosave complete E:\FLU\DataMart\JAN06\DEVEL\SAS Ln 1, Col 1

How I Did It

The power of CALL EXECUTE

- An example of the original code I moved into %SETUP from each single study-specific program looked like this (I had to manually specify all occasions when this code was needed):

```
%MACRO char2num(ds = , _var_ = , _len_ = );  
DATA &ds (DROP = _temp_);  
    LENGTH &_var_ 8;  
    SET &ds (RENAME = (&_var_ = _temp_));  
    &_var_ = INPUT(_temp_, &_len_.);  
RUN;  
%MEND;  
%char2num(ds = hospital, _var_ = page, _len_ = 3);
```

- The same code, now auto-sensing, after the variable manipulations were moved to %SETUP, looked like this:

```
* If there is a character PAGE or SERIES variable, make it numeric ;  
IF name IN ('PAGE' 'SERIES') & UPCASE(type) = 'CHAR' THEN CALL EXECUTE(COMPBL("  
DATA %SCAN(&&dset&i, 2) (DROP = temp_);  
    SET %SCAN(&&dset&i, 2) (RENAME = (" || name || " = temp_));  
    LENGTH " || name || " 8;  
    IF temp_ ^= '' THEN " || name || " = INPUT(COMPRESS(temp_), 2.);  
RUN;"));
```

How I Did It

Why you should use macro language

- To put it simply, using macro language was the only way I could maintain control over the application code once things started getting complex in terms of number of both studies and data domains
- I found especially useful the use of macro lists of items, like the &LIST and &LIST_PART ones referenced above, linked to the SAS-provided %WORDS macro and the %SCAN function
- In the paper you can find more details on a very tricky problem I had to face, entailing a shifting set of so-called post-injection reactions variables to which a ,quick box‘ had to be applied correctly after the pooling

How I Did It

Why you should use macro language: sample adult 'POSTINJ' CRF

Quick Box: If assessment is 0 for all reactions in a column, check Quick Box and leave rest of column blank.						Check box if no reactions below				If reaction was present post-vaccination on day 7, did it continue beyond post-vaccination day 7? See systemic codes below	1 = Yes 2 = No	If Yes, record AE#
						☐	☐	☐	☐			
INJECTION SITE	Ecchymosis (mm)									If present and measurable, enter LARGEST diameter; otherwise enter 0 (do not use decimals, > or < signs, or units of measurement). Enter "PLT" for reactions resulting in emergency room (ER) visit or hospitalization		
	Erythema (mm)											
	Induration (mm)											
	Swelling (mm)											
	Pain											
SYSTEMIC	Chills									0 = None 1 = Mild (transient with no limitation in normal daily activity) 2 = Moderate (some limitation in normal daily activity) 3 = Severe (unable to perform normal daily activity) 4 = PLT [emergency room (ER) visit or hospitalization]		
	Malaise											
	Myalgia											
	Arthralgia											
	Headache											
	Sweating											
	Fatigue											

How I Did It

Writing forward-looking code

- In my experience, the difference between a good and an average programmer can be measured by how they approach the programming problems they encounter
 - While an average programmer will tend to stick strictly to the parameters of the situation they are presented with, a good one will structure the code in a way that will make its later extension easier
 - The first will use a lot of IF-THEN-ELSE blocks, the second will rather use SELECT-OTHERWISE
 - Along the same lines, one will use 'EQ' while the other will use 'IN' (incidentally, it's unfortunate there is still no %IN macro function)
- Adopting more generalisable constructs is an investment, which will probably pay back nicely in the long run
- Use of macro language, general by definition, helps again

Agenda

Introduction

Overall Organization

The Final Goal: Standard Structures

Study Dimension

Data Domain Dimension

How I Did It

Moving from SASHELP to PROC SQL

The Power of CALL EXECUTE

Why You Should Use Macro Language

Writing Forward-looking Code

Conclusions

Conclusions

- The version of the SAS System you will use has a major impact on your code: SAS 9.2 can natively do things I had to program piece by piece with SAS 6.04 (just think about the ODS), so always have the manuals ready!
- Explore SASHELP thoroughly, but be ready to switch to PROC SQL for more speed (Be careful though, „Your mileage may vary“)
- With CALL EXECUTE you can run a (parameterised) DATA step in the middle of another one, and more
- Look at the big picture, and see if you can make your code more general without compromising its effectiveness

Conclusions (2)

- Take time to study the macro language, and experiment with it extensively: it will be a difficult and sometimes frustrating experience, but will help your programming skills grow to a new level
- Make good use of the availability of SAS resources on the web, starting from the manuals themselves on to the many good sites with plenty of tested code: if you are lucky you can use it to solve your problem, at worst you can always learn something new
- And remember, there are always multiple ways to do the same thing, so be ready to critically review your own code as your skills expand (or other people look at it)

References

- Online SAS manuals:
<http://support.sas.com/documentation/index.html>
- %WORDS macro: <http://support.sas.com/kb/26/152.html>
- SAS-L: <http://www.listserv.uga.edu/archives/sas-l.html>
- Roland's SAS Macros:
<http://www.datasavantconsulting.com/roland/>
- Wei Cheng's SAS links site: <http://www.prochelp.com/>
- PhUSE 2009 CALL EXECUTE presentation:
<http://www.phuse.eu/download.aspx?type=cms&docID=1414>

Question time

“Are you being served?”

