Paper AD11

Creating applications with mash-ups: user written functions, stored sql queries, and interfaces

Steve Prust, Elderbrook Solutions, Bietigheim-Bissingen, Germany

ABSTRACT

In the pharmaceutical development world the processes of clinical trial reporting, responding to regulator queries and so on it can be difficult to create applications. Combining techniques from the tools and concepts available to us can be a fruitful area of improving both speed of delivery and ease of development. This paper looks at "mashing-up" some of these suitable techniques.

INTRODUCTION

The paper looks at three suitable tools together with the concept of a data warehouse:

%GetSQL macro
ADaM and "one-step-away"
Data warehousing
SAS ® / AF application

Each tools / technique is discussed in detail below and where other tools may be switched in is also identified.

USER WRITTEN FUNCTIONS AND %GETSQL MACRO

User -written functions were introduced SAS v9 (initially limited but v9.2 made enhancements) They can be relatively simple, for example here is a user written function to calculate a celsius value from a fahrenheit value:

```
proc fcmp outlib=sasuser.mysubs.utility;
 function F2C(fahrenheit) ;
 return((fahrenheit-32)/1.8);
 endsub;
run;
```

and to use it:

```
options cmplib=sasuser.mysubs;
data test;
  do tempf = 32 to 102 by 9;
    tempc = f2c(tempf);
    put tempf= tempc=;
  end;
run;
```

with these results:

```
TEMPF=32 TEMPC=0
TEMPF=41 TEMPC=5
TEMPF=50 TEMPC=10
TEMPF=59 TEMPC=15
TEMPF=68 TEMPC=20
TEMPF=77 TEMPC=25
TEMPF=86 TEMPC=30
TEMPF=95 TEMPC=35
```

This looks simple enough and it is easy to consider SAS functions a poor relation to SAS Macros where the ability to process any steps is easy to envisage. This aspect of SAS functions was directly addressed in a paper to the SAS Global Forum in 2012 by the author Mike Rhoads:

> "*what do you do if you need a macro that can be embedded within a SAS statement, but the underlying task requires the execution of one or more DATA or PROC steps?"*

His solution was to develop a combination of macros and functions that allowed this to be done. The solution was of three parts:

- Outer macro – invokes a user written function
- User written function – a bridge between inner and outer macros

- Inner macro – does the work – can contain whatever sequence of PROCs and DATA steps needed

This paper included a macro - %GetSQL – that took this technique and so allowed complex functionality to be included wherever a function could be used in SAS (which gives a broad range of opportunities).

Here is the set of macros from the paper (see the paper itself for more details of how the techniques work) and then an example of its use.

```
%MACRO GetSQL / PARMBUFF;
 %let SYSPBUFF = %superq(SYSPBUFF);
 %let SYSPBUFF = %substr(&SYSPBUFF,2,%LENGTH(&SYSPBUFF) - 2);
 %let SYSPBUFF = %superq(SYSPBUFF);
 %let SYSPBUFF = %sysfunc(quote(&SYSPBUFF));
 %local UNIQUE_INDEX;
 %let UNIQUE_INDEX = &SYSINDEX;
 %sysfunc(GetSQL(&UNIQUE_INDEX,&SYSPBUFF))
%MEND GetSQL;

proc fcmp outlib=sasuser.mysubs.utility;
 function GetSQL(unique_index_2, query $) $ 32;
 length query query_arg $ 32000 viewname $ 32;
 query_arg = dequote(query);
 rc = run_macro('GetSQL_Inner', unique_index_2, query_arg, viewname);
 if rc = 0 then return(trim(viewname));
 else return('*** ERROR ***');
 endsub;
run;

%MACRO GetSQL_Inner;
 %local query;
 %let query = %superq(query_arg);
 %let query = %sysfunc(dequote(&query));
 %let viewname = GetSQL_tmpview_&UNIQUE_INDEX_2;
 proc sql;
 create view &viewname as &query;
 quit;
%MEND GetSQL_Inner;
```

Sample call:

```
options cmplib=sasuser.mysubs;
data example;
 set
 %GetSQL (select name, age, weight from sashelp.class where sex = 'M' order by
age)
 %GetSQL (select name, age, height from sashelp.class where sex = 'F' order by
name)
 ;
run;
proc print data=example;
run;
```

and the results:

he SAS System

Obs	NAME	AGE	WEIGHT	HEIGHT
1	Thomas	11	85.0	.
2	James	12	83.0	.
3	Robert	12	128.0	.
4	John	12	99.5	.
5	Jeffrey	13	84.0	.
6	Alfred	14	112.5	.
7	Henry	14	102.5	.
8	William	15	112.0	.
9	Ronald	15	133.0	.
10	Philip	16	150.0	.
11	Alice	13	.	56.5
12	Barbara	13	.	65.3
13	Carol	14	.	62.8
14	Jane	12	.	59.8
15	Janet	15	.	62.5
16	Joyce	11	.	51.3
17	Judy	14	.	64.3
18	Louise	12	.	56.3
19	Mary	15	.	66.5

ADAM DATASETS AND "ONE STEP AWAY"

The question of whether or not ADaM datasets in practice give us datasets that actually are one-step away from producing an analysis is not the principal point of this topic. In the author's experience there are many situations where the ADaMs available need some manipulation before analyses can be done.

However, by using ADaM structures and SQL the "one-step away" concept can be enabled.
For example an Adverse Events analysis involving frequencies, rate ratios / rate differences is not particularly easy to program is one step in SAS from a conventional ADAE dataset, but an SQL query of can facilitate this. Using the CDISC pilot data as an source (and this pilot study is used for all examples in this paper), this query:

```
proc sql noprint;
  create table aeanal_rr as
     select a.*, b.tottrtdur, b.toty, b.npt,
     (b.tottrtdur - a.offset) / 365.25  as tar,
     (100*nae*365.25/(b.tottrtdur - a.offset)) as rate,
     (100*nae/npt) as pctae
     from
       (
         select trtan, trta, aebodsys, aedecod, sum(offset) as offset, n(offset) as nae
         from
         (
           select distinct usubjid, astdy, (trtedt-trtsdt+1) as trtdurd, (trtedt-
trtsdt+1-astdy) as offset,
             trtan, trta, aesoc, aebodsys, aedecod, min(sub_a.astdy) as min_astdy
           from cdp.adae (where=(TRTEMFL='Y')) as sub_a
           group by usubjid, aesoc, aebodsys, aedecod
           having astdy = min_astdy
         )
         group by trtan, trta, aebodsys, aedecod
       ) as a
     inner join
       (
         select sum(trtdur) as tottrtdur, sum(trtdur)/365.25 as toty, n(trtdur) as npt,
trt01an, trt01a
           from cdp.adsl
           group by trt01an, trt01a
       )  as b
     on a.trtan = b.trt01an
     order by trtan, trta, aebodsys, aedecod
  ;
quit;
```

produces this dataset – which is suitable for various analyses (or direct reporting):

VIEWTABLE: Work.Aeanal_rr

	TRTAN	TRTA	AEBODSYS	AEDECOD	OFFSET	NAE	TOTTRTDUR	TOTY	NPT	TAR	RATE	PCTAE
1	0	Placebo	CARDIAC DISORDERS	ATRIAL FIBRILLATION	0	1	12820	35.099247091	86	35.099247091	2.8490639626	1.1627906977
2	0	Placebo	CARDIAC DISORDERS	ATRIAL HYPERTROPHY	168	1	12820	35.099247091	86	34.639288159	2.8868953525	1.1627906977
3	0	Placebo	CARDIAC DISORDERS	ATRIOVENTRICULAR BLOCK FIRST DEGREE	68	1	12820	35.099247091	86	34.913073238	2.8642565872	1.1627906977
4	0	Placebo	CARDIAC DISORDERS	ATRIOVENTRICULAR BLOCK SECOND DEGREE	6	1	12820	35.099247091	86	35.082819986	2.8503980022	1.1627906977
5	0	Placebo	CARDIAC DISORDERS	BRADYCARDIA	163	1	12820	35.099247091	86	34.652977413	2.8857549182	1.1627906977
6	0	Placebo	CARDIAC DISORDERS	BUNDLE BRANCH BLOCK LEFT	-1	1	12820	35.099247091	86	35.101984942	2.848841744	1.1627906977
7	0	Placebo	CARDIAC DISORDERS	BUNDLE BRANCH BLOCK RIGHT	143	1	12820	35.099247091	86	34.707734428	2.8812021772	1.1627906977
8	0	Placebo	CARDIAC DISORDERS	CARDIAC FAILURE CONGESTIVE	-1	1	12820	35.099247091	86	35.101984942	2.848841744	1.1627906977
9	0	Placebo	CARDIAC DISORDERS	MYOCARDIAL INFARCTION	165	4	12820	35.099247091	86	34.647501711	11.544843935	4.6511627907
10	0	Placebo	CARDIAC DISORDERS	SINUS ARRHYTHMIA	152	1	12820	35.099247091	86	34.683093771	2.8832491317	1.1627906977
11	0	Placebo	CARDIAC DISORDERS	SINUS BRADYCARDIA	66	2	12820	35.099247091	86	34.918548939	5.7276148659	2.3255813953
12	0	Placebo	CARDIAC DISORDERS	SUPRAVENTRICULAR EXTRASYSTOLES	134	1	12820	35.099247091	86	34.732375086	2.8791581271	1.1627906977
13	0	Placebo	CARDIAC DISORDERS	TACHYCARDIA	12	1	12820	35.099247091	86	35.066392882	2.8517332917	1.1627906977
14	0	Placebo	CARDIAC DISORDERS	VENTRICULAR HYPERTROPHY	14	1	12820	35.099247091	86	35.06091718	2.8521786663	1.1627906977

And this query is suitable for use in the %GetSQL macro:

```
data example2;
 set %GetSQL
 ( select a.*, b.tottrtdur, b.toty, b.npt,
      (b.tottrtdur - a.offset) / 365.25  as tar,
…
      on a.trtan = b.trt01an
      order by trtan, trta, aebodsys, aedecod);
run;
```

Log - (Untitled)

```
NOTE: The quoted string currently being processed has become more than 262 characters long.  You might have unbalanced quotation marks.
NOTE: The quoted string currently being processed has become more than 262 characters long.  You might have unbalanced quotation marks.
NOTE: The query requires remerging summary statistics back with the original data.

NOTE: SAS threaded sort was used.
NOTE: There were 1126 observations read from the data set CDP.ADAE.
      WHERE TRTEMFL='Y';
NOTE: There were 254 observations read from the data set CDP.ADSL.
NOTE: There were 354 observations read from the data set WORK.GETSQL_TMPVIEW_17.
NOTE: The data set WORK.EXAMPLE2 has 354 observations and 12 variables.
NOTE: DATA statement used (Total process time):
      real time           0.37 seconds
      cpu time            0.15 seconds
```

This query (and any other query) definition could also be stored, here. for example in a SAS macro variable:

```
%let _temp_query = select a.*, b.tottrtdur, b.toty, b.npt,
     (b.tottrtdur - a.offset) / 365.25  as tar,
     (100*nae*365.25/(b.tottrtdur - a.offset)) as rate,…
```

And used thus:

```
proc print data=%GetSQL (&_temp_query) ; run;
```

```
The SAS System

                                                A
                                                E                     A
                                                B                     E          O
     T                                          O                     D          F
     R       T                                  D                     E          F
 O   T       R                                  S                     C          S
 b   A       T                                  Y                     O          E
 s   N       A                                  S                     D          T

326 81 Xanomeline High Dose SKIN AND SUBCUTANEOUS TISSUE DISORDERS PRURITUS GENERALISED    146
327 54 Xanomeline Low Dose  SKIN AND SUBCUTANEOUS TISSUE DISORDERS RASH                    800
328  0 Placebo              SKIN AND SUBCUTANEOUS TISSUE DISORDERS RASH                    212
329 81 Xanomeline High Dose SKIN AND SUBCUTANEOUS TISSUE DISORDERS RASH                    466
330 54 Xanomeline Low Dose  SKIN AND SUBCUTANEOUS TISSUE DISORDERS RASH ERYTHEMATOUS         4
331 81 Xanomeline High Dose SKIN AND SUBCUTANEOUS TISSUE DISORDERS RASH MACULO-PAPULAR      182
```

Which starts to look simpler to use and offers some flexibility.

DATA WAREHOUSE

CONCEPT

In pharmaceutical programming we have many definitions as to the data intended for analysis. These are generally supplied in statistical analysis plans of one sort or another. Assuming that these definitions are well constructed (complete, unambiguous and so on) then the data selections themselves should be able to be defined as an SQL query. Effectively these can become the basis of a data warehouse.

The concept of a data warehouse is more procedural than technical in its implications. Data selections need careful specification, codifying and validation. These are not onerous tasks but once they are done they simplify and accelerate other parts of the process. For example if an efficacy analysis is based on a data specification involving various covariates and many potential subgroups then this needs a clear means of specifying what combinations are allowable or what constraints may exist. Having done this specification and validated the query definition we have – with methods such as %GetSQL - a means to have validated data selections directly available.

This availability of validated data selections has the ability to speed up investigative analyses and general reporting.

STORAGE

The definitions that form the data warehouse need to be stored somehow. Aside from aspects such as validation, change control, scope and so on a basic consideration is the storage format. For the purposes of this mash-up JSON was used as it has good flexibility and can be interfaced by most software.

PUTTING IT TOGETHER

With a data warehouse in place and a tool such as %GetSQL we are close to a means of accessing and processing data quickly, reliably and aligned with Good Programming Practice (GPP). We need something now to stitch these components together: an application of some sort.

APPLICATION

CHOICE OF APPLCATION ENVIRONMENT

The requirements of an application for this mash-up are relatively straightforward:

- Access the definitions of stored queries in the data warehouse definitions.
- allow the queries to be modified appropriately (modifications that are within the constraints defined for each specific query)
    - this might include selecting a subset or a subgroup, or defining a subgroup based on – for example - demographic / baseline characteristics.
- access the data using the query (with or without modification)
- provide access to analysis techniques with the selected data, this includes:
    - procedures and routines from a software package such as R or SAS,
    - and/or programmed macros that perform a standard analysis.

For reasons of ease of use with %GetSQL developer familiarity with the software, and access to standard analysis / reporting macros SAS A/F was chosen for the application environment. SAS A/F is an object-orientated programming environment with a simple and effective programming language (SCL) that has many similarities to SAS and can used most SAS functionality.

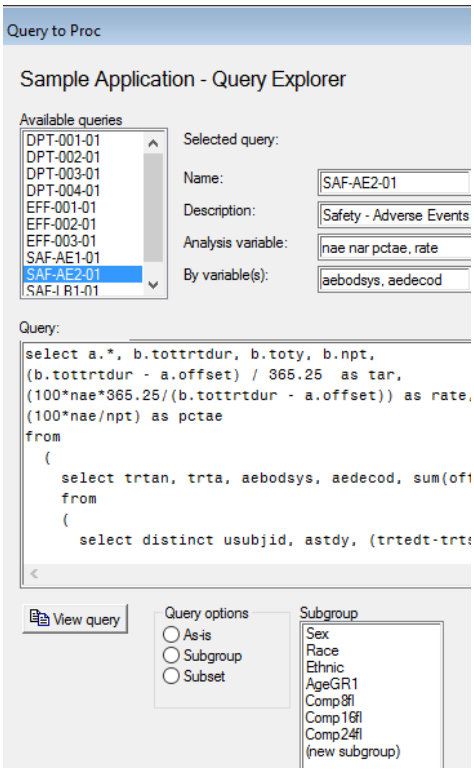
SAMPLE APPLICATION

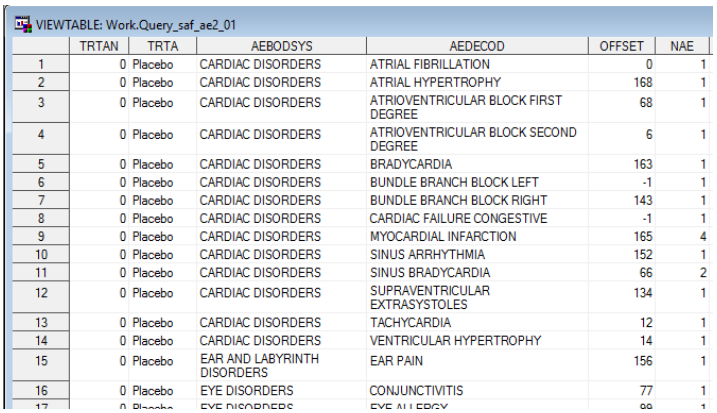
The sample application is based on a single screen with some dialog boxes to support data choices. The processing is:

- read the data warehouse query definitions and make available for selection
- enable modifications of the query (for example the definition of a new subgroup is via a dialog box that allows the user to interactively specify data selections)
- process the query and either
    - make data available for viewing
    - perform an appropriate analysis (with interactive setting of options)
- lastly, there is the ability to review and store the resultant analysis (together with the generating code)

Below are some screen shots showing the application in action firstly with same AE data and then with some vital signs data:

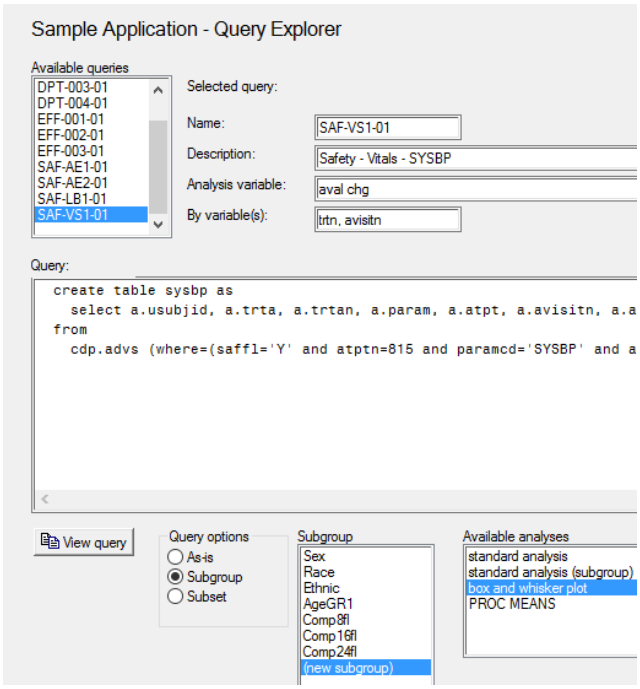
- The main screen showing the query being selected, processed, and passed to the VIEWTABLE window (the *View query* button has been pressed in the left screenshot):



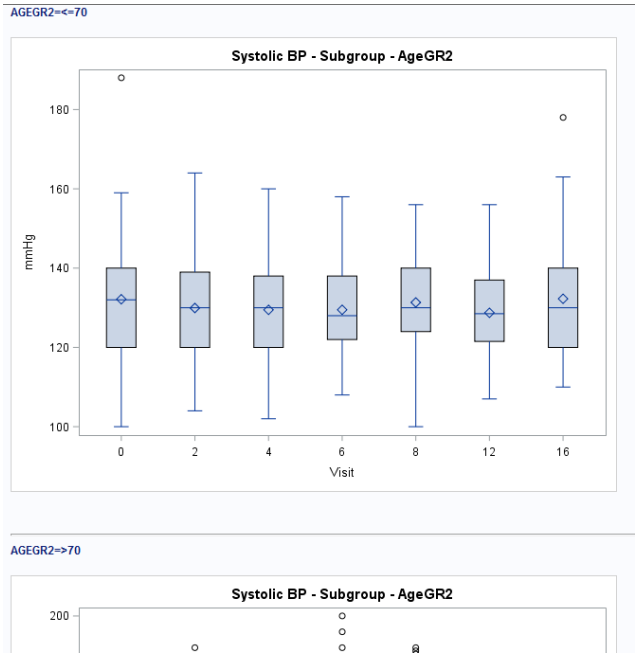


	TRTAN	TRTA	AEBODSYS	AEDECOD	OFFSET	NAE
1	0	Placebo	CARDIAC DISORDERS	ATRIAL FIBRILLATION	0	1
2	0	Placebo	CARDIAC DISORDERS	ATRIAL HYPERTROPHY	168	1
3	0	Placebo	CARDIAC DISORDERS	ATRIOVENTRICULAR BLOCK FIRST DEGREE	68	1
4	0	Placebo	CARDIAC DISORDERS	ATRIOVENTRICULAR BLOCK SECOND DEGREE	6	1
5	0	Placebo	CARDIAC DISORDERS	BRADYCARDIA	163	1
6	0	Placebo	CARDIAC DISORDERS	BUNDLE BRANCH BLOCK LEFT	-1	1
7	0	Placebo	CARDIAC DISORDERS	BUNDLE BRANCH BLOCK RIGHT	143	1
8	0	Placebo	CARDIAC DISORDERS	CARDIAC FAILURE CONGESTIVE	-1	1
9	0	Placebo	CARDIAC DISORDERS	MYOCARDIAL INFARCTION	165	4
10	0	Placebo	CARDIAC DISORDERS	SINUS ARRHYTHMIA	152	1
11	0	Placebo	CARDIAC DISORDERS	SINUS BRADYCARDIA	66	2
12	0	Placebo	CARDIAC DISORDERS	SUPRAVENTRICULAR EXTRASYSTOLES	134	1
13	0	Placebo	CARDIAC DISORDERS	TACHYCARDIA	12	1
14	0	Placebo	CARDIAC DISORDERS	VENTRICULAR HYPERTROPHY	14	1
15	0	Placebo	EAR AND LABYRINTH DISORDERS	EAR PAIN	156	1
16	0	Placebo	EYE DISORDERS	CONJUNCTIVITIS	77	1

- In the screenshots below a new subgroup has been defined (this is by means of a dialog box and is not shown here) and then a box and whisker plot has been requested:



And here is the resultant box and whisker plot using a new subgroup:



- Finally, some of the code behind the frame (the View query button) showing some SCL plus the %GetSQL macro call (the section between `submit continue`; and `endsubmit`):

```
pbViewQuery:
  _temp_qname = tranwrd(teSelQName.text,'-','_');
  _temp_qbvar = teSelQbvar.text;
  _temp_query = left(getitemc(tpSelQuery.text,1));
  do i = 2 to listlen(tpSelQuery.text);
    _temp_query = trim(_temp_query) || ' ' || left(getitemc(tpSelQuery.text,i));
  end;
  _temp_query = trim(_temp_query) || ' order by ' || trim( _temp_qbvar) ;
  submit continue;
    data work.tq_&_temp_qname ;
      set %GetSQL (&_temp_query) ;
    run;
  endsubmit;
  call execcmd('VIEWTABLE work.tq_' || trim(_temp_qname) );
RETURN;
```

CONCLUSION

This mash-up is to show an example of what is possible. It provides a fast and reliable method of performing new data analyses. The potential for distributing it to users outside the programming department is considered possible and such distribution would probably provide some useful tools during situations where investigative analyses are being sought.

There are several aspects of the mash-up that could be improved and/or done differently. The purpose of this paper however is not to provide a perfect solution but to prompt others to consider how they might mash things together to produce new and better tools and solutions.

REFERENCES

Use the Full Power of SAS in Your Function-Style Macros, Mike Rhoads, Paper 004-2012, SAS Global Forum 2012

CONTACT INFORMATION

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

Stephen Prust
Elderbrook Solutions Gmbh
Borsigstraße 4, 74321 Bietigheim-Bissingen, Germany
Work Phone: +49 7142 339 29 19
Email:stephen.prust@elderbrooksolutions.com