Paper 118-2013

A Day in the Life of Data - Part 3

Peter Crawford, Crawford Software Consultancy Limited

ABSTRACT

As a new SAS® programmer, you may be overwhelmed with the variety of tricks and techniques that you see from experienced SAS programmers; as you try to piece together some of these techniques you get frustrated and perhaps confused because the data showing these techniques are inconsistent. That is, you read several papers and each uses different data. This series of four papers is different. They will step you through several techniques but all four papers will be using the same data. The authors will show how value is added to the data at each of the four major steps: Input, Data Manipulation, Data and Program Management, and Graphics and Reporting.

INTRODUCTION

This paper lies among the Foundations and Fundamentals stream offering a view of a day in the life of data. Having “input the data” in the first of these four papers and “manipulated the data” in the second until it seems just fine, before turning to the final step which analyzes the data presenting results and insight, this paper introduces a pause. (I always pause before trying to imagine what management means) – to consider the management of programs and data.

OK, if you have been following this stream, you'll understand that there are some programs and data prepared and possibly (or probably knowing those programmers we see here today) they will be fit to re-use, so we should plan and clarify how these provide a “service” to the SAS user who is at the center of the final stage in this “day in the life of data“.

OVER-RIDING OBJECTIVE

The customers, who have invested in all these programmers and all those software resources and the collecting of all that data, are hoping for analyses that provide useful insight.

It is the analyst, or data miner, who will be tasked to produce this insight. To have a chance of achieving success, the analyst must concentrate on developing that insight. That job will be a lot easier if the analyst:

1. does not need to be the programmer who knows all about the input data and their sources and the data cleansing needed;
2. is not the developer who has to know all about the manipulation needed to organize all that data into data marts that are focused on diverse business views while remaining entirely consistent;
3. has easy access to drill down into any and all aspects of the data on which he will report.

PURPOSE OF PAPER

To show how, by good management of programs and data

- the data miner or analyst should have easy and confident access to the underlying data,
- all processes are auditable,
- any continuing need to run data loading programs should no longer require the original program developer

STEPS TO ACHIEVE GOOD MANAGEMENT OF PROGRAMS AND DATA

STEP 1 – SHARE

Even when there is only one programmer involved in all stages of producing an analysis, the structure described by these four papers in “A day in the life of Data” provides a useful model for subdividing the entire challenge. In the more normal situation there must be a sharing of access to data among a team, and a lot of benefits come when the same attitude applies to the sharing of program management. It is very unlikely that two developers write the same program, however, each might take advantage of components created by the other.

Not all code needs to be rewritten for each report and analysis. Some code will be easily reused. Place that code in a folder that can be read by all developers and all programs that might take advantage of that code. Programs are easily invoked with syntax like:

```
%INCLUDE "&shared_folder/popular_program.sas" ;
```

You can prepare a pointer to the folder like:

```
FILENAME shrcode "&shared_folder" ;
```

This filename statement needs to run just once in the SAS session, and it remains available until replaced or the SAS session ends.

Then you can reuse the code with syntax like:

```
%INCLUDE shrcode( popular_program ) ;
```

When providing code that is to be shared, it is good professional practice to use the operating system to mark these programs as "read only" - just to prevent accidents ;-)

The first and perhaps most significant benefit coming from this approach is that your program becomes shareable far more quickly because, when passed to another developer of the team or to an analyst, only the SHRCODE reference needs to change. Supporting this, many development teams have the policy that physical paths, like the value which is in macro variable &SHARED_FOLDER, should be removed from programs and only the logical references like SHRCODE and library references (like WORK and SASHELP) should appear. They keep the setting of these logical references to folders or files separately. Some teams restrict the setting of physical paths to metadata, others to the AUTOEXEC for the SAS job. A less common solution uses operating system environment variables. An example of this appears in all SAS sessions – These use environment variables, at least for SASROOT and SASAUTOS. They also use environment variables to point into the SAMPLE library - try using library SAMPSIO or including sample code from SAMPSRC.

A mixture of techniques should be seen as normal. However, the presence of physical paths in programs should be seen only as a temporary solution during program development.

Another idea encouraged by good management and supported by the above techniques: all analysts and developers should expect to use the same LIBNAME references for the data marts.

I know that there will be occasions when a single LIBNAME is not enough for all users of the DATA area. For example, when a report wants to compare data in two generations of an ordinary data mart, the typical LIBNAME for the current data mart is no longer enough to reference an earlier generation. In that case the LIBREF might usefully incorporate some kind of generation number or even a date. This provides an alternate LIBNAME not a substitute for a simple definition.

SHARE MACROS

We can also share the convenience of SAS language macros.

There are great benefits that come when using macros that are common among our teams. For a start everyone will know what your code will do when it invokes %something if this is shared and in common usage.

Separate these macros into a shared folder that can be used with the SASAUTOS option. It is a reliable way of shared use and the rules are straightforward.

RULES OF SASAUTOS

To be able just to invoke a macro without first pre-compiling it, the SAS session needs to be able to locate where it is stored. The SASAUTOS option provides a list of places for the SAS session to search. Three definitions are possible:

1. a FILEREF (the default is SASAUTOS which usually points to more than one folder)
2. a physical path to the folder holding macro code
3. in parentheses (), a list of logical and / or physical references

The third option, the list, imposes order. Once the code for a macro is found the search ends, the macro is compiled, and that version will be used.

So the next “RULEs OF SASAUTOS” pay attention to that imposed order::

4. When creating a new macro, store it in a folder you APPEND to the SASAUTOS value list. This ensures your new macro will be proved to safely co-exist with the usual macro collection.
5. When testing a change to an existing macro, you need the "AUTOCALL macro search" to find your test version before the standard, so store the test macro in a macro development folder, and INSERT the path to that development folder into the SASAUTOS option (INSERT goes at the front).
6. The Analyst can also take advantage of the SASAUTOS facility when using macros.

These rules allow a developer, or someone who is challenged to fix a problem or provide changes, more easily to prepare an update without needing to replicate the entire library.

```
OPTION INSERT =SASAUTOS "&developers_macs" ;
```

Of course that doesn't impose the change for all users but it does enable seamless testing. Placing this statement in a shared AUTOEXEC file could impose it.

```
OPTION APPEND =SASAUTOS "&additional_macs" ;
```

This will extend an existing library of macros but not override those already available.

These APPEND/INSERT features might be unfamiliar to seasoned SAS developers with many years of experience, because they only became effective (in an OPTION statement) in SAS release 9.2. Since SAS release 9.3, the following code lists system options which can be affected by INSERT or APPEND.

```
PROC OPTIONS  LISTINSERTAPPEND ;
RUN;
```

The results appear in the SASlog, like

```
4    proc options  listinsertappend define value; run;

     SAS (r) Proprietary Software Release 9.3  TS1M1


     Core options that can utilize INSERT and APPEND

      AUTOEXEC           Identifies AUTOEXEC files used during initialization
      CMPLIB             Identify previously compiled libraries of CMP subroutines to use
                         when linking
      FMTSEARCH          List of catalogs to search for formats and informats
      MAPS               Location of maps for use with SAS/GRAPH
      SASAUTOS           Search list for autocall macros
      SASHELP            Location of the SASHELP library
      SASSCRIPT          Location of SAS/CONNECT script files

     Host options that can utilize INSERT and APPEND

      HELPLOC            Location of help environment text and index files
      MSG                The path to the SAS MSG directory
      SET                Defines an environment variable

NOTE: PROCEDURE OPTIONS used
```

Output 1. Output from PROC OPTIONS feature LISTINSERTAPPEND

This feature of the OPTIONS procedure is not valid in SAS9.2. However, the OPTION statements above, with INSERT and APPEND, are effective in the SAS9.2 release. Earlier releases supported INSERT and APPEND options but only on the command line or in a configuration file and not on a SAS program statement.

By now becoming available on statements, this new OPTION functionality removes the need for my macro, %ADDOPTN(), which was designed to add to, rather than replace, the SASAUTOS value. It is still demonstrated on the SAS Support website at http://support.sas.com/kb/24/841.html in the SAS Sample 24841: Executing Your Stored Formats at Start-Up The macro was very helpful for extending the reach of the SASAUTOS option into new macro libraries and the FMTSEARCH option to new formats.

My final “RULE OF SASAUTOS”

1. Store each macro in its own file named with the name of the macro

MANAGING DATA SETS

Basic information about the structure and size of data sets can be obtained from the utility SAS procedures CONTENTS and DATASETS.

```
PROC CONTENTS DATA= orion.product_list VARNUM ;
RUN ;
```

Which is almost the same as

```
PROC DATASETS LIBRARY= orion        NOLIST ;
     CONTENTS    DATA= product_list VARNUM ;
RUN ;
QUIT ;
```

When managing a collection of data in one or more libraries, you might also want to associate additional information like owner, update frequency, program name or program suite that creates it - perhaps also a date when you no longer need or wish to keep it. Rather than the CONTENTS statement or PROC you might discover the convenience in the SQL dictionary tables. The one which holds information about all the columns in all the tables in all the libraries can quickly and efficiently provide just the libraries (and optionally, tables) you choose.

```
TITLE    'orion product_list data structure' ;
FOOTNOTE "report at %now( fmt= twmdy )" ;
PROC SQL ;
   SELECT name, label, type, length, format, varnum, idxusage, sortedby
     FROM sashelp.vcolumn
    WHERE libname = 'ORION'
      AND memname = 'PRODUCT_LIST'
 ORDER BY libname, memname, varnum
        ;
QUIT ;
```

The SAS9.3 Results Viewer shows

orion product_list data structure

Column Name	Column Label	Column Type	Column Length	Column Format	Column Number in Table	Column Index Type	Order in Key Sequence
Product_ID	Product ID	num	8	12.	1	SIMPLE	1
Product_Name	Product Name	char	45		2		0
Supplier_ID	Supplier ID	num	4	12.	3		0
Product_Level	Product Level	num	3	12.	4	SIMPLE	0
Product_Ref_ID	Product Reference ID	num	8	12.	5		0

report at 14:17 Monday, February 25, 2013

Table 1

When seeking a particular column among many tables SASHELP.VCOLUMN makes a great resource.

> Of course when handling such a lot of stuff from a SAS environment there would be something you must remember: Although you can, ***don't*** use SASHELP views in a data step or through anything but PROC SQL for access to these larger dictionary tables – instead **always** use PROC SQL – and your queries will complete very much faster.

For an often repeated query, create your own SQL view. I use such a view almost more often than daily, keeping an eye on the limited free space available in my working Teradata databases.

When you want a complete table of the columns in all tables, the CONTENTS procedure / statement OUT= option, must execute once for each library you manage and still you will need PROC SQL to add your table or column information. (and if SQL is not your choice you'll be invoking DATA STEP, PROC SORT etc to avoid one SQL join.

SQL is well worth learning.

WHAT WE'RE HERE FOR

We extract information from multiple libraries:

– to provide a look-up list of all columns throughout the application

– as a resource for all users of the application – in particular:

- the Analyst community (who might add extra information)
- any developer who has to maintain the application (we don't want inadvertent re-use of a column name)
- any user who might need to trace where a column is created

WE'RE HERE ABOUT MANAGING PROGRAMS AND DATA.

Managing need not mean being the expert on all you manage. You just need a quick and easy way to show where all the programs are and what purpose each serves and, even more, the same for data.

In fact the manager I am describing will be really useful even knowing nearly nothing about data analysis, data manipulation or even about data loading – just so long as this manager can provide quick, easy, reliable access to the world of information about the application.

That solution requires collecting information from SAS, but need not be limited to a SAS platform for display. I am not the first to find Microsoft excel provides a convenient surface for lists. I am sure there are or will be suitable alternatives. My point is that this layer is about information more than process.

COLLECTING INFORMATION ABOUT PROCESSES

A very neat trick I saw demonstrated at an earlier SUGI or SASGF- have standardized doc headers in your code – see appendix 1- these would allow a program to collect purpose, author, suite, and frequency : like

```
Data parse1 ;
  infile &in("*.sas") /* &in holds the FILEREF with all the code */
       scanover column=col filename= filen ;
  input @"&seek"  @ ; * &SEEK holds the text to search for, like purpose: ;
  *make sure the file ends .SAS (*.SAS7BDAT is a SAS data set)  ;
  if upcase(scan( filen, -1, '.' )) ne 'SAS' then delete ;
  purpose = substr( _infile_, col) ; *taking the rest of the line ;
  prog = filen ; * vars named in statement options cannot go into output DS ;
```

will collect a piece of metadata from each program in the SHRCODE folder, in an automated fashion. Use code like this to collect from each code folder and for each relevant piece of metadata, and more, subject to your standards.

The SAS enhanced editor – the program editor in SAS Enterprise Guide® provides a facility referred to as "abbreviations" that can insert a standard program header. The "keyboard macros" allow these abbreviations to be extended, inserting current date and time information.

Collecting one piece of metadata at a time might seem less efficient than collecting several in a single pass, but with the SAS/Graph Sample library as a model, 137 programs provide title, name, reference, keys in sub-second run-times with the code in Appendix 2.

CONCLUSION

- Know What's What!

A little SQL, a little information and you will find a lot of help from SAS.

Would you like "impact analysis" like from what tables the info has come? And how old that data might be? That's all about managing programs and data. The ideas above provide the concepts for collecting the building blocks. Want to know the processes? Then spend little longer on this paper and presentation. All code will become available after SAS Global Forum on the www.sascommunity.org website.

To request the Orion and Orion Gold data used in this paper, send an email to dayinlife@fernwood.ca to obtain the download address.

REFERENCES

PAPERS ON PARSING SAS CODE

Probably there are a great many papers which might improve on these, so, after reviewing the three below, make further research at ww.lexjansen.com or your preferred search engine.

1. www.nesug.org/Proceedings/nesug11/cc/cc19.pdf An Animated Guide: A program to Read information from Program Headers and Produce Reports for Programming Managers, Russ Lavery, Contractor
2. www2.sas.com/proceedings/sugi25/25/cc/25p074.pdf An Automated Method to Create a Descriptive Index for a Directory of SAS Programs by Gary Cunningham, Sanofi-Synthelabo Research, Malvern, PA
3. www.nesug.org/proceedings/nesug04/po/po04.pdf Creating an HTML Index of an Autocall Library from Standardized Macro Headers: Experiences from the American College of Radiology Imaging Network, by Quentin McMullen of Brown University Center for Statistical Sciences

STANDARD PROGRAM HEADERS

4. An Animated Guide: A Program to Read Information in SAS Headers www.nesug.org/Proceedings/nesug11/cc/cc19.pdf (Parsing SAS code 1. above)
5. SAS Code Documentation, http://www.datasavantconsulting.com/roland/document.html

ACKNOWLEDGMENTS

The excellent help from Section Chairs Peter Eberhardt and Harry Droogendyk.

RECOMMENDED READING

- *See the papers on parsing SAS code above*
- *Base SAS® Procedures Guide*
- *SAS Companion to the platforms that you use.*

CONTACT INFORMATION

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

Name:	Peter Crawford
Enterprise:	Crawford Software Consultancy limited
Address:	London, UK
E-mail:	Peter.Crawford@blueyonder.co.uk

APPENDIX

1 PROGRAM HEADER EXAMPLE

```
/*------------------------------------------------------------------------*
*   purpose    : one-line description for the program                    *
*   program    : physical file name                                      *
*   version    : 0.0 is development draft                                *
*   author     : original author see later for updates                   *
*   suite      : application context                                     *
*   frequency  : ad-hoc or daily/weekly/monthly                          *
*  prerequisites                                                         *
*   parameters : list or general description of parameters               *
*   read tables: list dbase.table1                                       *
*   read files : /pathname/subfolder/text                                *
*  outputs                                                               *
*   tables     : list tables written                                     *
*   files      : list files written                                      *
*   reports    : generic report number/ description                      *
*  changes_____________________________________________________________*
*   date : who : action                                                  *
*-------------------------------------------------------------------------*/
```

2 COLLECT METADATA FROM SAS SAMPLE LIBRARY

```
%MACRO search_progs( seek             /* always require the string to find*/
                   , in  =shrcode     /* path to programs                 */
                   , out =parse_&seek /* output table to hold results     */
                   )/ DES='collect some info' ;
DATA &out  ;
  LENGTH purpose prog filen $1024 ;
  INFILE &in("*.sas") /* &in holds the FILEREF with all the code */
       SCANOVER COLUMN= col FILENAME= filen ;
  INPUT @"&seek"  @ ; * the text to search for, like purpose: ;
  *check the file ends '.SAS' (and not SAS7BDAT = SAS data set)  ;
  IF UPCASE(SCAN( filen, -1, '.' )) NE 'SAS' THEN DELETE ;
  purpose = SUBSTR( _INFILE_, col) ; *taking the rest of the line ;
  prog = filen ; * vars named in statement options cannot go into output DS ;
  /* remove trailing *, |,  and trailing block comment  from purpose */
  IF SUBSTR( purpose, LENGTH(purpose)-1) = '*/' THEN
     SUBSTR( purpose, LENGTH(purpose)-1) = ' ' ;
  IF SUBSTR( purpose, LENGTH(purpose),1) IN('*', '|') THEN
     SUBSTR( purpose, LENGTH(purpose)  ) = ' ' ;
  prog = filen ; * vars named in statement options cannot go into output DS ;
RUN ;
%MEND  search_progs ;

FILENAME gsample '!sasroot\graph\sample' LRECL=1024 ;

%search_progs( seek= %str( KEYS:)   /* always require the string to find*/
             , in=gsample           /* path to programs                 */
             , out= parse_keys      /* to hold results                  */
                    ) ;
%search_progs( seek= %str( REF:)    , in=gsample, out= parse_ref  ) ;
%search_progs( seek= %str( NAME:)   , in=gsample, out= parse_name ) ;
%search_progs( seek= %str( TITLE:)  , in=gsample, out= parse titl ) ;
%search_progs( seek= %str( PRODUCT:), in=gsample, out= parse_prod ) ;
* now bring the pieces together ;
PROC SQL ;
 CREATE TABLE code_meta AS
       SELECT a.purpose AS program
            , b.purpose AS title
            , c.purpose AS product
            , D.purpose AS keys
            , COALESCE( a.prog, b.prog, c.prog, d.prog) as progn
         FROM parse_name      a
    FULL JOIN parse_titl      b  ON b.prog = a.prog
    FULL JOIN parse_prod      c  ON c.prog = a.prog
```

```
    FULL JOIN parse_keys      d  ON d.prog = a.prog
  ORDER BY  progn
      ;
QUIT ;
```

A small part of the output table WORK.CODE_META

program	title	product	keys
GABMUMVR	GABNUMVR-Area Bar Chart with a Numeric Chart Variable	GRAPH	GRAPHICS ODS GAREABAR
GABSUBGR	GABSUBGR - Generating an Area Bar Chart with Subgroups	GRAPH	GRAPHICS ODS GAREABAR
GABSUMVR	GABSUMVR - Generating an Area Bar Chart	GRAPH	GRAPHICS ODS GAREABAR
GABWSTAT	GABWSTAT-Generating an Area Bar Chart	GRAPH	GRAPHICS ODS GAREABAR
GANCIRCL	GANCIRCL-Drawing a Circle of Stars	GRAPH	GRAPHICS ANNOTATE GSLIDE MACRO
GANCITY	GANCITY-Labeling Cities on a Map	GRAPH	GRAPHICS ANNOTATE PATTERN GMAP
GANMULTI	GANMULTI-Using NAME= to Produce Multiple Graphs	GRAPH	GRAPHICS ANNOTATE GANNO
GANSCALE	GANSCALE-Scaling Data-Dependent Output	GRAPH	GRAPHICS ANNOTATE GANNO
GANSQUAR	GANSQUAR-Storing Annotate Graphics	GRAPH	GRAPHICS ANNOTATE GANNO
GANVBAR	GANVBAR-Labeling Subgroups in a Vertical Bar Chart	GRAPH	GRAPHICS AXIS PATTERN ANNOTATE GCHART