Structured Approach Using MACRO Code

Allen J. Blackburn, Foreign Trade Division, US Bureau of Census

ABSTRACT

If you're still looking for another really clever way or trick to use that SAS macro, you don't really need this. But, suppose for instance that you've got that semi small or semi large staff filled with bright journeymen, or novice SAS programmers. And suppose that you have to get that Balance Of Import and Export Trade Press Release produced this month and every month thereafter. Your monthly task should you decide to accept is to process millions of records each month arriving in similar text format from various brokers, carriers, miscellaneous businesses government agencies, and even governments. Each year new features, legislation, and programs require us to tweak and manipulate, yet maintain a basic data structure to incorporate yearly, even monthly changes throughout a seemingly large mountain of SAS programs. Our division is charged with the maintenance of the integrity of a data structure across large numbers of SAS datasteps, PROC Sorts, PROC Tabulates, and others including PROC Means.

The SAS MACRO allows us the ability to %INCLUDE code which is "EXTRA TO" or "OUTSIDE" the basic code module. Our proc's code basically defines all text fields with name, length, and type to be "INPUT"ed to all programs. Control information also is created via breakpoint information to document numbers of inputs and outputs and other value control information thereby enabling us to determine correctness and create permanent records. The SAS MACRO is transparent to all levels of programmer ability. By using sameness or oneness of the approach in programming development, our programmers have injected a bit of the 30 year-old Structured Programming concept. Further, it provides self documentation. Even further it provides one set of complete, tested, correct code over the entire program(mer) base.

INTRODUCTION

Suppose that you have to get that Balance Of Import and Export Trade Press Release produced this month and every month thereafter. Trade arrives from multiple sources via myriad methods of transportation. Even paper is received which requires data entry. Our division maintains a data structure. Our project scope does not radically or quickly change; we need to keep online with the same processing system time frames and of course programmed in SAS. We process multiple production runs each week which can vary statistical month, both past, current and future. Major project scope changes at the back end occur with the use of SAS MDDB tools combined with html code for data distribution over internet. That topic is also being presented at this conference by Mr. Blake Sanders titled *GoodsHound - Building Multi-Functional Web-Based applications with SAS/IntrNet® and JavaScript* (Application Development #AD03).

Some History

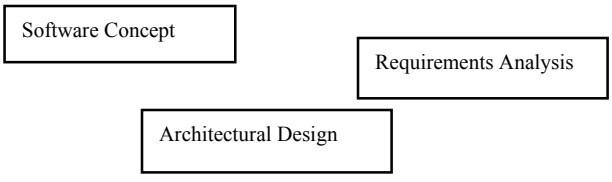
We had traveled down 25 years of COBOL batch programming road, “SORT - MERGE - TABULATE - PRINT”, “SORT - AGAIN MERGE - AGAIN TABULATE - AGAIN PRINT - AGAIN”. SAS offered us new technology. At first SAS was something oriented toward the heavy-duty research, data analysis and geek speak. When in the course of these human events it became advantageous because of terabyte storage media coupled with seemingly endless processor speeds, we moved to consider the high end SAS use.

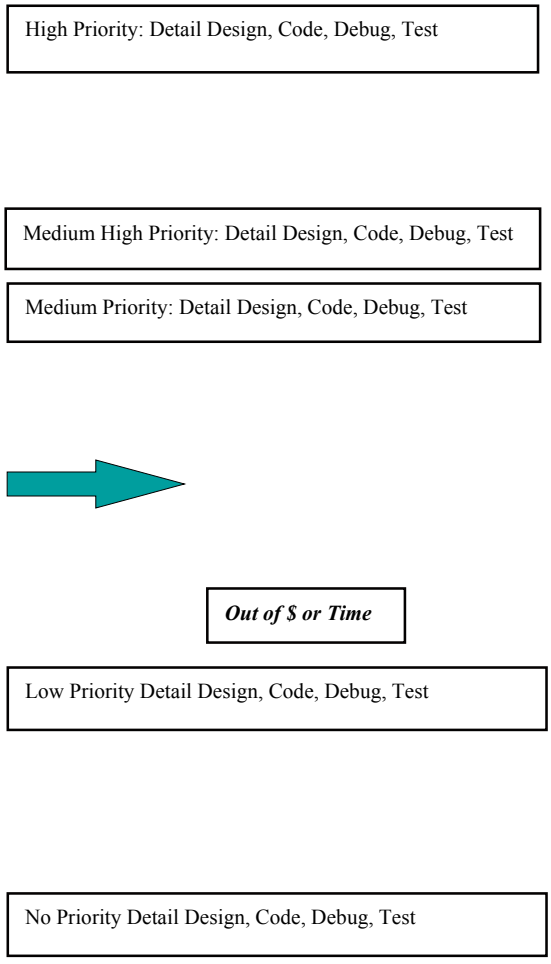
At first the focus was to use the tool independently. Learn it. Get comfortable with it. Propagate it. PROC MEANS, PROC TABULATE, PROC FREQ. Do the analysis, and create fun graphs. But for the long run, our job was to do the grunge I/O, reformat data, sort it, merge it, and test the data against known parameters to determine pass or fail of a complex edit against very fine value sets identifying outliers, in the end reformat the data and finally pass it through PROC's TABULATE and MEANS. Our job is the dirty word **batch** programming with a capital "**B**".

Naturally every COBOL program and system in place required reprogramming. Long time COBOL programmers that we were, we sought to bring to SAS the same structure and its self-documentation. As much as 100% of our monthly batch processing is parameter-driven.

We could emulate the COBOL with the SAS easily enough. And the SAS was fast enough. We had enough storage capacity. But the COBOL/SAS emulation needed a little work. To be honest, SAS can be too productive, too easy sometimes. Our specifications were already in place. This was a functioning system. Our Life Cycle Model selection could be termed "Design to Schedule", as described in Steve McConnell's *Rapid Development*. Any "Evolutionary" approach with its give and take, ease of change, and total development flexibility, would have been disastrous with our rigid deliverables schedule, which is often driven by legislation and inter government MOU. Using the "design to schedule" approach allowed us to bring components online as they were completed often ahead of schedule. A following diagram is from Mr. McConnell's *Rapid Development* chapter 7, "Life Cycle Planning".

Design To Schedule:





We required a structured approach. And we needed team time to think it through. Design time. And old programmers were being replaced by new people, an eclectic and diverse group. The workforce was competent in SAS either through experience or in the process of being SAS-trained. The best and brightest set about to define all necessary sets of ASCII inputs and outputs, reduce variable naming to a standard, provide control count mechanisms, handle errors, and define "LIBNAME" library locations. McConnell states that "planned reuse offers the greatest schedule and effort savings more than any other rapid development practice". In 1973, when speaking of OS/360 development, Frederick Brooks described this

%INCLUDE process, using PL/1, as “Direct Incorporation” of “intermodule interfaces”. Perhaps 20-25% savings on the overall project life could be realized through “reuse code” according to McConnell.

Some technical considerations involving Reuse:

1. Focus on domain specific details - If you’re working on a financial application, focus on building reusable financial components.
2. Create small sharp components.
3. Create good documentation
4. Make it error-free.
5. Focus on quality, not size.
6. Don’t worry about developers using the reusable code.

An example: libnames: simple but effective with one overall definition.

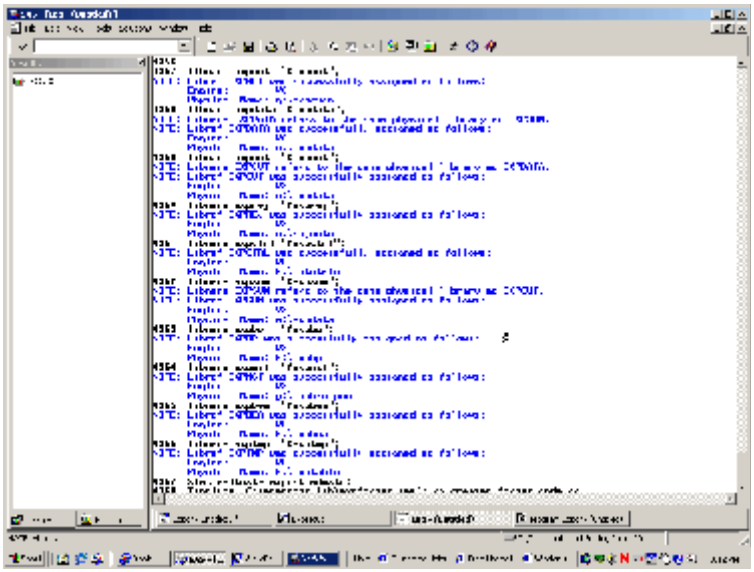
c:\expdata\location.fil :

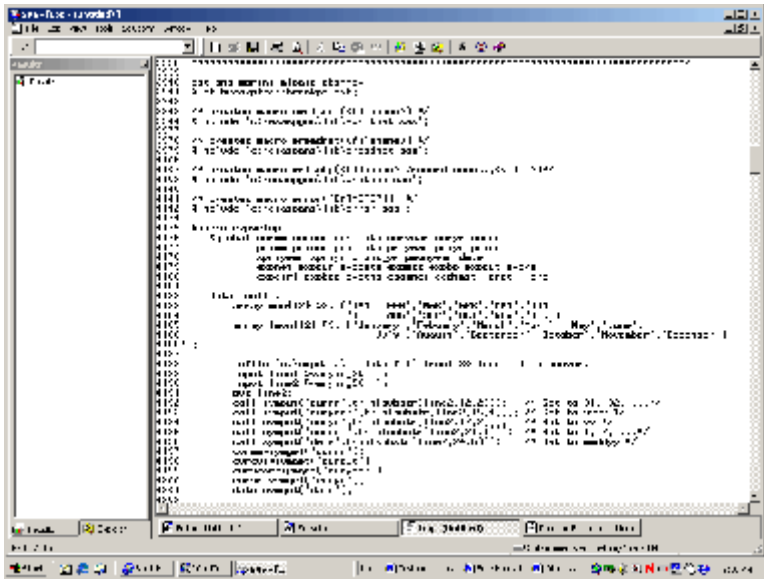
```
OUT e:\expdata
CTL E:\cutctrls
CUT E:\expdata
REJ e:\rejects
NET g:\expeven
SUM e:\expdata
MAS g:\cspb\expcut
BEA E:\expbea
BP  E:\expbp
TMP E:\exptable
```

```
data _null_;
     infile 'c:\expdata\location.fil' lrecl=30
        length=ln missover;
      input line1 $varying30. ln;
      if upcase(substr(line1,1,3)) = 'OUT' then do;
           call symput('expdata',trim(substr(line1,5)));
           expdata=symget('expdata');
           put 'EXPDATA: ' expdata;
      end;
      if upcase(substr(line1,1,3)) = 'SUM' then do;
           call symput('expsum',trim(substr(line1,5)));
           expsum=symget('expsum');
           put 'EXPSUM: ' expsum;
      end;
      if upcase(substr(line1,1,3)) = 'CUT' then do;
           call
symput('expcut',trim(substr(line1,5)));
           expcut=symget('expcut');
           put 'EXPCUT: ' expcut;
      end;
      if upcase(substr(line1,1,3)) = 'REJ' then do;
           call
symput('exprej',trim(substr(line1,5)));
           exprej=symget('exprej');
           put 'EXPREJ: ' exprej;
      end;
      if upcase(substr(line1,1,3)) = 'NET' then do;
          call symput('expnet',trim(substr(line1,5)));
expnet=symget('expnet');
           put 'EXPNET: ' expnet;
      end;
      if upcase(substr(line1,1,3)) = 'MAS' then do;
          call symput('expmst',trim(substr(line1,5)));
expmst=symget('expmst');
           put 'EXPMST: ' expmst;
      end;
      if upcase(substr(line1,1,3)) = 'BEA then do;
          call symput('expbea',trim(substr(line1,5)));
expbea=symget('expbea');
           put 'EXPBEA: ' expbea;
      end;
      if upcase(substr(line1,1,3)) = 'TMP then do;
          call symput('exptmp',trim(substr(line1,5)));
exptmp=symget('exptmp');
           put 'EXPTMP: ' exptmp;
      end;
      if upcase(substr(line1,1,3)) = 'CTL' then do;
         call symput('expctrl',trim(substr(line1,5)));
expctrl =symget('expctrl');
           put 'EXPCTRL: ' expctrl;
      end;
      if upcase(substr(line1,1,2)) = 'BP' then do;
           call symput('expbp',trim(substr(line1,5)));
expbp=symget('expbp');
           put 'EXPBP: ' expbp;
      end;
      if _error_=1 then %error(DATASTEP);
   run;
```

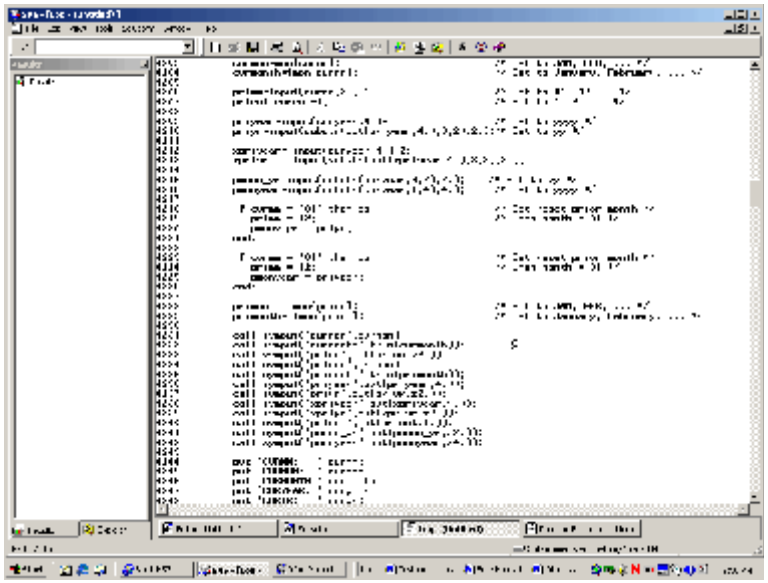
and for reference, in the log:

```
EXPDATA: e:\expdata
EXPSUM: e:\expdata
EXPCUT: E:\expdata
EXPREJ: e:\rejects
EXPNET: g:\expeven
EXPMST: g:\cspb\expcut
EXPBEA: E:\expbea
EXPTMP: E:\exptable
EXPCTRL: E:\cutctrls
EXPBP: E:\expbp
```





Macros have been created in libname statement form:

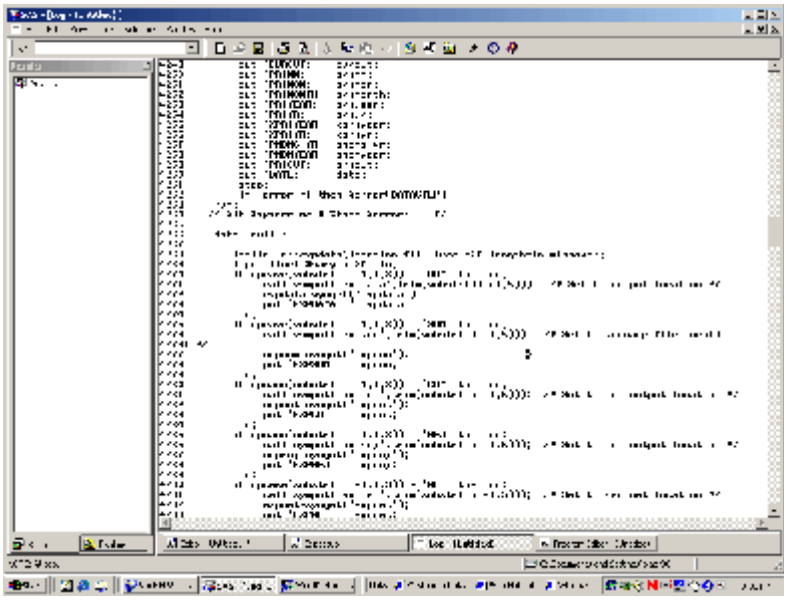


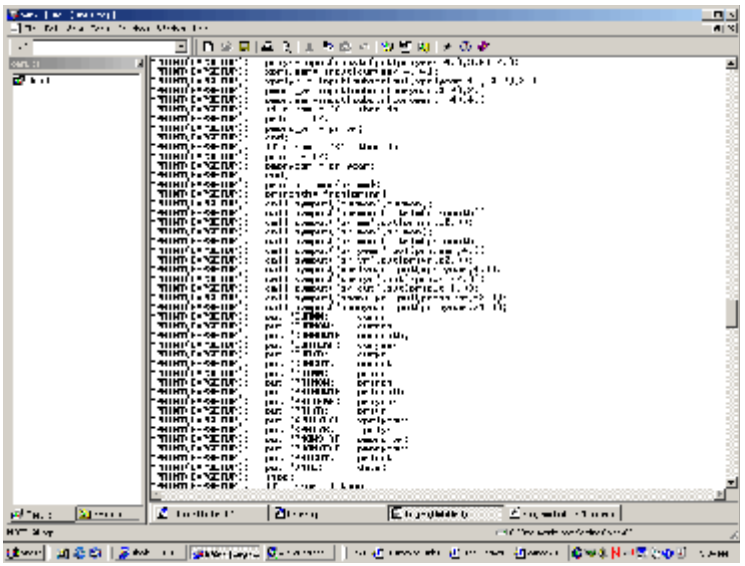
Definitions of month, current month, prior month, future month, current year, prior year, and future year as well. What we were to discover and hopefully circumvent was what Frederick Brooks' described as *The Mythical Man-Month* concept, taming development time somewhat before we run out of time into unrecoverable tar pits. Programmers would not be set upon SAS coding without some aid or bridge to structured programming. The MACRO was that bridge. We termed it our "SETUP".

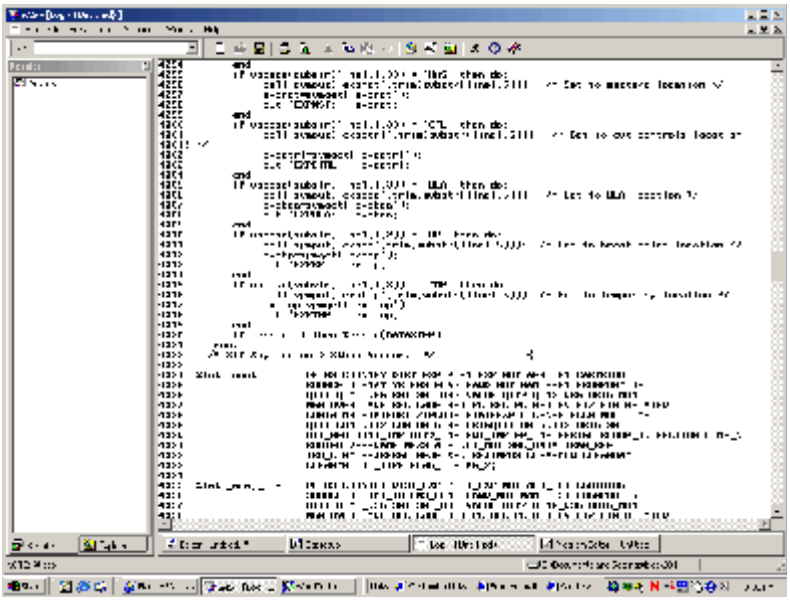
c:\expdata\adddata.fil – expsetup reads an adddata test file to initially setup the month and year codes in varying styles as well as several other setup procedures
HEADER ***MON MM YYYY CUT* DATE*****
EXPLAB MAR 03 2002 3 042902
MON - Statistical Month abbreviation (JAN,FEB,...,DEC)
MM - Statistical Month Number (01,02,....,12)

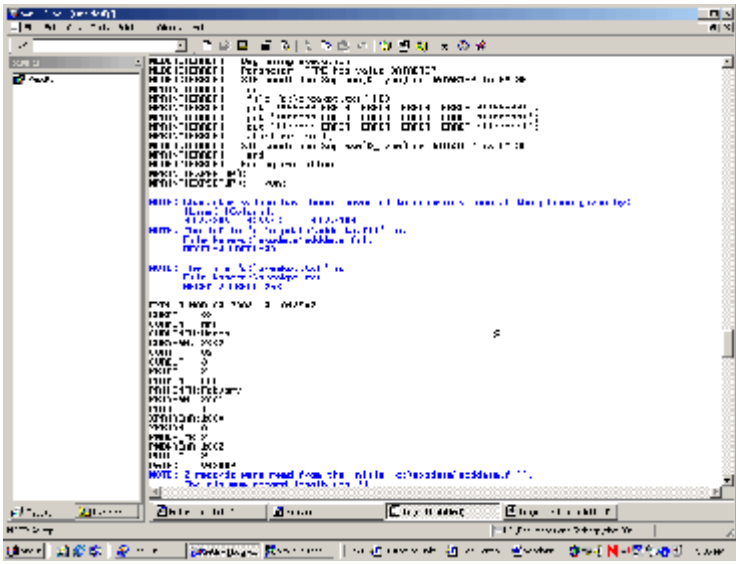
YYYY - Statistical Year
CUT* refers to a sequential numbered batch run in one statistical month 1,2,..5
DATE* refers to today

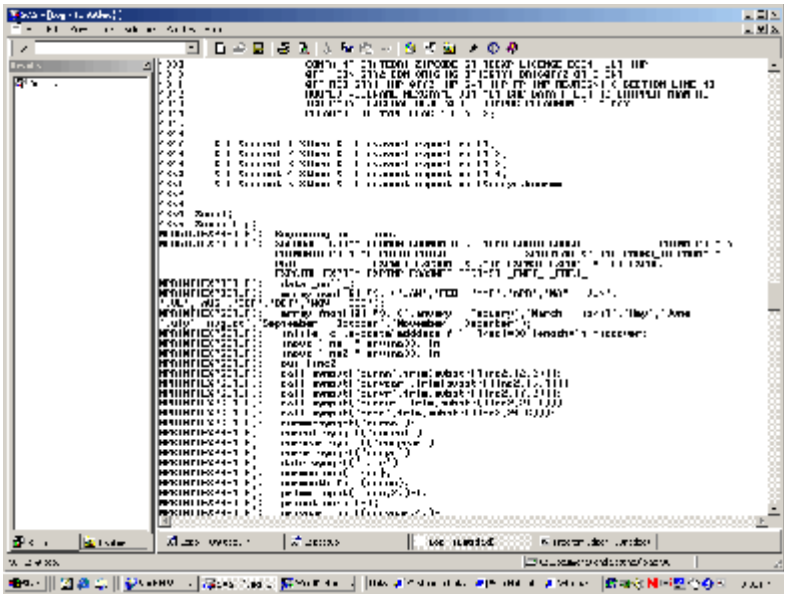
Multiple cuts are run during the statistical month during the statistical year with data accessible for analysis and publication approximately 45 days after goods reach port.











The base structure, the NET was defined using our current structure, since it had not failed us in many years. Modification from the COBOL to SAS produced most similar names and attributes. The maintenance of this base structure assure integrity of the entire system. That integrity lies not only with *this* processing year but all of our historical files as well.

Summary:

In summary, the SAS Macro offers up a potential for creating and using standardized coding modules to bring up programming staffs up to speed. I have only described a couple, but there many that could help your offices. This is certainly not new. Frederick Brooks in the early 70's and through following decades with Steve McConnell, into the present, using a structured approach can save you time and money in development costs.

References: Brooks, Frederick P. Jr., *The Mythical Man-Month*, Addison-Wesley Publishing Company, 1973.

McConnell, Steve, *Rapid Development*, Microsoft Press, 1996.

Contact Information:

Allen J. Blackburn
301-457-2216
US Bureau of Census
Foreign Trade Division
allen.j.blackburn@census.gov