Leave Your Bad Code Behind: 50 Ways to Make Your SAS® Code Execute More Efficiently.

William E Benjamin Jr
Owl Computer Consultancy, LLC

Topic Groups

- Processing more than one file in each DATA step
- Combining steps to make simple tasks take less code
- Using Macro variables to simplify maintenance
- Using built in features rather than your own code
- Ways to save disk space
- Using sorts for more than just sorting data
- Ways to make the program code just read better
- Advanced coding techniques that are hard to find
- A Macro routine that will conditionally execute Code

Processing more than one file in each DATA step

- Tip 1 – Make a temporary dataset and change a variable name:

Two Passes Over The Data	One Pass Over The Data

```
Data temp_file_1;
        Set Perm.input_file;
Run;

Data temp_file_2;
        Set temp_file_1;
        Rename var_1 = var_a;
Run;
```

```
Data temp_file_2;
      Set Perm.input_file
            (Rename var_1=var_a);
Run;


(var_a is available in the data step)
```

Processing more than one file in each DATA step

- Tip 2 – Make a temporary dataset and change a variable name:

Two Passes Over The Data	One Pass Over The Data

```
Data temp_file_1;
        Set Perm.input_file;
Run;

Data temp_file_2;
        Set temp_file_1;
        Rename var_1 = var_a;
Run;
```

```
Data temp_file_2
        (Rename var_1 = var_a);

    Set Perm.input_file;
Run;


(var_1 is available in the data step)
```

Processing more than one file in each DATA step

- Tip 3 – Make two similar temporary dataset copies from the same input file:

Two Passes Over The Data	One Pass Over The Data

```
Data temp_file_1;
        Set Perm.input_file;
Run;

Data temp_file_2;
        Set Perm.input_file;
Run;
```

```
Data    temp_file_1
        temp_file_2;

        Set Perm.input_file;
Run;
```

Processing more than one file in each DATA step

- Tip 4 – Make two different temporary dataset copies from the same input file:

Two Passes Over The Data	One Pass Over The Data

```
Data temp_file_1;
            Set Perm.input_file;
            If a then output;
Run;

Data temp_file_2;
            Set Perm.input_file;
            If b then output;
Run;
```

```
Data        temp_file_1
            temp_file_2;

      Set Perm.input_file;
       If a then output temp_file_1;
       If b then output temp_file_2;
Run;
```

Processing more than one file in each DATA step

- Tip 5 – Read and Merge Two External Files: (Slide 1)

Two Passes Over The Data

```
filename one_file 'G:\SGF_2012\file1';
filename two_file 'G:\SGF_2012\file2';

data read_one;
      infile one_file linesize=160 truncover;
      input @01 text $char80. @81 more_data $char80. ;

data read_two;
      infile two_file linesize=160 truncover;
      input @01 text $char80. @81 more_data $char80. ;
run;

data merged;
set read_one  read_two;
Run;
```

Processing more than one file in each DATA step

- Tip 5 – Read and merge two external files: (Slide 2)

One Pass Over The Data

```
filename two_file ('G:\SGF_2012\file1'
                   'G:\SGF_2012\file2');

data merged;
      infile two_file linesize=160 truncover;
      input @01 text $char80. @81 more_data $char80. ;
run;
```

Processing more than one file in each DATA step

- Tip 6 – Interleave data from two External Files: (Slide 1)

Two Passes Over The Data --- Output Dataset is Interleaved

```
filename one_file 'G:\SGF_2012\file1';
filename two_file 'G:\SGF_2012\file2';

data read_one;
        infile one_file linesize=160 truncover;
        input @01 text $char80.      @81 more_data $char80. ;

data read_two;
        infile two_file linesize=160 truncover;
        input @01 text $char80.      @81 more_data $char80. ;

data merged;
set read_one; output;
set read_two; output;
Run;
```

Processing more than one file in each DATA step

- Tip 6 – Interleave data from two External Files: (Slide 2)

One Pass Over The Data --- Output Dataset is Interleaved

```
filename one_file 'G:\SGF_2012\file1';
filename two_file 'G:\SGF_2012\file2';

data read_two;
  infile one_file linesize=160 truncover;
  input @01 text      $char80.          @81 more_data $char80.;
  Output;

  infile two_file linesize=160 truncover;
  input @01 text      $char80.          @81 more_data $char80.;
  Output;
run;
```

Processing more than one file in each DATA step

- Tip 7 – Combine data from two External Files: (Slide 1)

Two Passes Over The Data – Data is sequentially added to output file

```
filename one_file 'G:\SGF_2012\file1';
filename two_file 'G:\SGF_2012\file2';

data read_one;
        infile one_file linesize=160 truncover;
        input @01 text $char80.      @81 more_data $char80. ;

data read_two;
        infile two_file linesize=160 truncover;
        input @01 text $char80.      @81 more_data $char80. ;

data merged;
set read_one  read_two;
Run;
```

Processing more than one file in each DATA step

- Tip 7 – Combine data from two External Files: (Slide 2)

One Pass Over The Data – Data is sequentially added to output file

```
filename one_file 'G:\SGF_2012\file1';       filename two_file 'G:\SGF_2012\file2';

data read_two;
      infile one_file linesize=160 truncover end=eof1;
            do until(eof1);
                  input @01 text $char80.      @81 more_data $char40.;
                  output;
            end;
      infile two_file linesize=160 truncover end=eof2;
            do until(eof2);
                  input @01 text $char80.     @81 diff_data $char80.;
                  output;
            end;
Run;
```

Combining steps to make simple tasks take less code

- Tip 14 – Reading text data into an array:

Calculate every Data Location	Calculate only first Array Location

```
Filename text 'D:\SGF_2012\Text_file';

Data file_with_an_array;

      Array myvars{3} $ 25 var_1-var_3;

      Infile text linesize=100 truncover;
      Input @01 key $char25.
            @26 var_1 $char25.
            @51 var_2 $char25.
            @76 var_3 $char25.;
Run;
```

```
Filename text 'D:\SGF_2012\Text_file';

Data file_with_an_array;

      Array myvars{3} $ 25 var_1-var_3;

      Infile text linesize=100 truncover;
      Input @01 key $char25.
            @26 (var_1-var_3) ($char25.);
Run;

(SAS calculates starting locations)
```

Combining steps to make simple tasks take less code

- Tip 15 – Reading complex data into SAS arrays:

(Two Passes Over The Data)	(One Pass Over The Data)

```
Filename text ‘D:\SGF_2012\Text_file’;

Data temp;

Array myvars1{3} $ 10 vara_1 vara_2 vara_3;
Array myvars2{3} $ 15 varb_1 varb_2 varb_3;

Infile text linesize=85 truncover;
   Input @01 key $char10.
   @11 vara_1 $char10. @21 varb_1 $char15.
   @36 vara_2 $char10. @46 varb_2 $char15.
   @61 vara_3 $char10. @71 varb_3 $char15.
    ;
Run;
```

```
Filename text 'f:\SGF_2012\Text_file.txt';

Data temp;

Array myvars1{3} $ 10 vara_1 vara_2 vara_3;
Array myvars2{3} $ 15 varb_1 varb_2 varb_3;

Infile text linesize=85 truncover;
Input @01 key $char10.
   @11 (vara_1-vara_3) ($char10. +15)
   @21 (varb_1-varb_3) ($char15. +10);
Run;
```

What if the array had 1000 elements?

```
input @01 key $char10.
   @11 (vara_1-vara_1000) ($char10. +15)
   @21 (varb_1-varb_1000) ($char15. +10);
```

Combining steps to make simple tasks take less code

- Tip 19 – Less Code to do the Same Work:

A Compare and Two Commands	One step to do the Same Work

Many programmers use the following code to test a variable and set an indicator to either 0 or 1:

```
Data _null_;
   Set Perm.input_file;
   If region = 'Africa'
     then flag_1 = 1;
     else flag_1 = 0;
Run;
```

user_cpu time used = 10.67 sec for 50 Million tests.

This code does the same thing:

```
Data _null_;
   Set Perm.input_file;
   flag_1 = region = 'Africa';
Run;
```

user_cpu time used = 10.47 sec for 50 Million tests.

Using Macro variables to simplify maintenance

- Tip 23 – Macro Variables to Control File names

Each name has to be updated	One change corrects all files

Note each reference needs to be changed if the directory ever changes.

```
Libname file_01
'c:\SGF_2012\project_dir\input_data';

Libname file_02
'c:\SGF_2012\project_dir\output_data';

Filename in_file1
'c:\SGF_2012\project_dir\my_text.txt';
```

Note the Double Quote will cause the macro variable to be resolved, and one change is applied to all three commands.

```
%let My_Dir = c:\SGF_2012\project_dir;

Libname file_01 "&My_dir.\input_data";

Libname file_02 "&My_dir.\output_data";

Filename in_file1 "&My_dir.\my_test.txt";
```

Using Macro variables to simplify maintenance

- Tip 24 – Control Array Parameters in One Place:

Each name has to be updated	One change corrects all References

Here three changes are needed if the array size changes.

```
Data array_test;
Array counters {15} var_01-var_15;

Do I = 1 to 15;
   Counters(i) = 0;
End;
```

Note here the macro variable allows the updating of the code in one place to change the array size and usage.

```
%let max = 15;

Data array_test;
Array counters {&max} var_01-var_&max;

Do I = 1 to &max;
  Counters(i) = 0;
End;
```

Using Macro variables to simplify maintenance

Tip 25 – Control code Using Macro Variables:

Each name has to be updated	One change corrects all References

If your programming environment includes more than one computer platform and operating system (OS) the file definition code (LIBNAME statements) can be coded into the program and comments (/* … */ or *…;) placed around the code needed for the other OS.

```
/* comment out the unused statement  */

*LIBNAME data1 '/unix/directory/name';

 LIBNAME data1 'd:\windows\directory\name';
```

SAS Automatic Macro variable SYSSCPL can be used to determine the O/S in use.

```
%let My_os = &SYSSCPL;

%assign_libs;
    %if &My_os = SunOS %then %do;
        LIBNAME data1 '/unix/directory/name';
  %end;
    %if &My_os = XP_PRO %then %do;
       LIBNAME data1 'd:\windows\directory\name';
  %end;
%mend assign_libs;
%assign_libs;
```

Using built in features rather than your own code

- Tip 30 – Building Arrays of Working Variables:

Write Code to Retain Variables	Use Built-in Features not Extra Code

If the array values need to survive more than one iteration of a DATA step then the variables need to be retained.

```
Data test;
  Array counter {3} var1-var3;
  Retain var1 var2 var3 0;

  Set my_data;      By key1;

  Counter(1) = Counter(1) + 1;
  Counter(2) = sum(var4,var5);
  Counter(3) = var6 * 3;

  If last.key1 then do I = 1 to 3;
     Counter(i) = 0;
  End;
  Drop i var1 var2 var3;
```

The SAS ARRAY _TEMPORARY_ option has the ability to reduce the code needed to do the same tasks.

```
Data test;

  Array counter {3} _temporary_;

  Set my_data;      By key1;
  Counter(1) = Counter(1) + 1;
  Counter(2) = sum(var4,var5);
  Counter(3) = var6 * 3;

  If last.key1 then do I = 1 to 3;
     Counter(i) = 0;
  End;

  Drop i;
```

Using built in features rather than your own code

- Tip 31 – Use Functions to Save Code:

Test all possible values	Test Without Knowing All Values

Instead of coding for all possible options of test cases in character strings like this:

```
if (a = 'YES' or
    a = 'YEs' or
    a = 'YeS' or
    a = 'yES' or
    a = 'yeS' or
    a = 'yEs' or
    a = 'Yes' or
    a = 'yes' ) then x = 1;
```

Use the SAS functions to compensate for minor variations in the variable formatting. The upcase function does not change the value of the variable in the following code, but it tests for all of the combinations of 'YES' in the column on the left.

```
if (upcase(a) = 'YES') then x = 1;
```

Using built in features rather than your own code

- Tip 34 – Score a Test Without Extra Variables:

The Extra Variable Counter is Created	No Extra Variables

To test if a score is more than 50% "Y" test if each of the flags are set to true.

```
Counter = 0;
If (Q1 = 'Y') then counter +1;
If (Q2 = 'Y') then counter +1;
If (Q3 = 'Y') then counter +1;
If (Q4 = 'Y') then counter +1;
If (Q5 = 'Y') then counter +1;
If (Q6 = 'Y') then counter +1;
If (Q7 = 'Y') then counter +1;
If (Q8 = 'Y') then counter +1;
If (Q9 = 'Y') then counter +1;
If ((counter/ 9) > .5) then Passed=1;
                      Else Passed=0;
```

The same result can be achieved without the use of a temporary counter variable as in the following code.

```
Passed = sum((Q1 = 'Y'),  (Q2 = 'Y'),
             (Q3 = 'Y'), (Q4 = 'Y'),
             (Q5 = 'Y'), (Q6 = 'Y'),
             (Q7 = 'Y'), (Q8 = 'Y'),
             (Q9 = 'Y')  )/9 > .5;
```

The result is a binary value (0 or 1) that is derived by adding the binary results of nine tests (Q1 = 'Y' …) and dividing the result by 9. Then comparing that result to .5 as in the code on the left with 0 = False, 1 = True.

Ways to save disk space

- Tip 36 – Make the SAS Files Smaller:

The standard installation	User Options available

The standard installation may not activate compression options for disk usage. As a result creating a SAS Dataset does not usually save disk space.

The SAS Option COMPRESS= has two options that produce output files that are smaller than the default option of COMPRESS=NO.

COMPRESS=YES - Produces a smaller SAS Dataset

COMPRESS=BINARY – Produces the smallest files. Some files can be reduced in size by 80% or more.

Both types of compressed files can be read without an "uncompress" step.

Ways to save disk space

- Tip 37 – Use a System Utility to Reduce Size:

The standard installation	User Options available
Use System Utilities or Operating system commands to compress files.	On Windows the Program Winzip can be used to compress files. But the files need to be “uncompressed” for SAS to be able to read them. On UNIX the gzip command can be used to compress files. But the files need to be “uncompressed” for SAS to be able to read them. The newer versions of Winzip on Windows can read and unzip the gzip files copied from Unix to Windows.

Ways to save disk space

- Tip 39 – Change the Size of Numeric Variables:

The standard installation	User Options available

Use the SAS ATTRIBUTE command to change the size of numeric variables, but remember that the maximum size of the number is smaller if the size of the variable is smaller.

Command Syntax =

ATTRIB var1 LENGTH = n;

Where n is an integer in the range of 3 to 8.

NOTE ** This also reduces the size of the number that can be stored in this variable !

Using sorts for more than just sorting data

- Tip 44 – Use Proc Sort for Subsetting too:

Two Passes Over The Data – Data is sequentially added to output file

Use the sort routine to combine any or all of the last three tips to subset your records at the same time as it is sorted and remove duplicates at the same time.

```
PROC SORT data=Temp_file(keep   =(key_1 var_1 var_2 var_3 var_4 var_5)
                         Rename  =(var_1=var_a var_2=var_b)
                           Where  =(var_1='A' and var_2='B'))
out=new_file nodupkey;
By key_1;
RUN;
```

***** Note *** The output file will have 6 variables (key_1 var_a var_b var_3 var_4 var_5) where var_1='A' and var_2='B'.**

Ways to make the program code just read better

- Tip 46 – If … Then … Else VS. The Select Clause: (Slide 1)

Cascading If … Then … Else Clauses can be hard to read !

At times a cascading "IF…THEN …ELSE" series of clauses can become hard to read, when indented over and over again, the SELECT statement structure operates in the same way as the "IF…THEN …ELSE", but is easier to read. Note the next slide:

```
IF var_a = 'A'
        then var_1 = 1;
        else if var_a = 'B'
                then var_1 = 2;
                else if var_a = 'C'
                        then var_1 = 3;
                        else var_1 = 4;
```

Ways to make the program code just read better

- Tip 46 – If … Then … Else VS. The Select Clause: (Slide 2)

Select Statements can be aligned easier to be more readable

Consider this instead:

```
Select(var_a);
        When('A') var_1 = 1;
        When('B') var_1 = 2;
        When('C') var_1 = 3;
 Otherwise        var_1 = 4;
 End;
```

Both these code segments run in about the same time, the SAS interpreter generates nearly the same code for both.

Advanced coding techniques that are hard to find

- Tip 48 – Bit testing to Combine multiple tests: (Slide 1)

Set Up One flag field with Many Conditions Represented:

```
Data Master_file;
   flag = '00000000000000000000000000000000';   * 31 zeros;

   Set my_sas_file;

      If (cond_1 = 1) then substr(flag,31,1) = '1';
      If (cond_2 = 2) then substr(flag,30,1) = '1';
     *. . . more conditions . . . ;
      If (cond_30 = 5) then substr(flag,2,1) = '1';
     If (cond_31 = 6) then substr(flag,1,1) = '1';
      Conditions_flag = input(flag,ib4.) ; * convert the flags to a real numeric variable;
                                           * IB4. informat is Integer Binary for 4 bytes.;
                                           * It knows there were 31 binary digits not 32;
 run;
```

Advanced coding techniques that are hard to find

- Tip 48 – Bit testing to Combine multiple tests: (Slide 2)

Subset the Data File by Picking Only the Flags You Need Today:

```
proc sort data = Master_file
          (where=(Conditions_flag=input('10100000000000000000000000000010',ib4.)))
              Out = New_subset_file;
  by key;
  run;
```

If the INPUT function on the WHERE= clause is not used (just the bit constant) then the Log will tell you that Bit constants are not compatible with the WHERE= clause. But the INPUT function converts the constant to a number before the WHERE= clause sees the constant. This way one master file can be used and subsets of data can be extracted for any proc that supports the SAS Dataset option WHERE=, like PROC FREQ, PROC PRINT, PROC SORT, and in the WHERE clause in a SAS DATA step.

A Macro routine that will conditionally execute Code

- Tip 50 – Build Code to Execute in Segments: (Slide 1)

Build 'Check-Pointed' Code Segments For Long Running Jobs :

By setting the flags to anything except YES the steps are conditionally executed.

```
*************************************************************************************;
** code Execution flags – YES will execute – .YES will not                         **;
*************************************************************************************;
%let extract_data_files     = .YES; * Flag to run the Extract data macro    **; *not run;
%let process_data_step1 = .YES; * Flag to run the Process step 1        **; *not run;
%let process_data_step2 = .YES; * Flag to run the Process step 2        **; *not run;
%let process_data_step3 =  YES; * Flag to run the Process step 3        **; * runs;
*****************************************************************;
* USER libname is a reserved libname to send SAS files to the "USER" Libname area;
Libname USER 'c:\SGF_2012\Work_area_that_is retained';
```

A Macro routine that will conditionally execute Code

- Tip 50 – Build Code to Execute in Segments: (Slide 2)

Build Each Job Step as a Separate Macro:

```
%macro Conditional_Step_A;
 ... SAS code to extract the data ...
%mend Conditional_Step_A;

%macro Conditional_Step_B;
 ... SAS code to process the data for Step 1 ...
%mend Conditional_Step_B;

%macro Conditional_Step_C;
 ... SAS code to process the data for Step 2 ...
%mend Conditional_Step_C;

%macro Conditional_Step_D;
 ... SAS code to process the data for Step 3 ...
%mend Conditional_Step_D;
```

A Macro routine that will conditionally execute Code

- Tip 50 – Build Code to Execute in Segments: (Slide 3)

Control the Execution of your Job Steps One at a Time:

```
%macro execute;
  %if &extract_data_files  = YES %then %do;
      %Conditional_Step_A;
  %end;

  %if &process_data_step1 = YES %then %do;
      %Conditional_Step_B;
  %end;

  %if &process_data_step2 = YES %then %do;
      %Conditional_Step_C;
  %end;

  %if &process_data_step3 = YES %then %do;
      %Conditional_Step_D;
  %end;

 %mend execute;
 %execute;
run;
```

Leave Your Bad Code Behind: 50 Ways to Make Your SAS® Code Execute More Efficiently.

QUESTIONS????

William E Benjamin Jr
Owl Computer Consultancy, LLC