

## **Are Your SAS® Programs Running You?**

Marje Fecht, Prowerk Consulting, Cape Coral, FL

Larry Stewart, SAS Institute Inc., Cary, NC

### **ABSTRACT**

Most programs are written on a tight schedule, using the most accessible knowledge of the programmer. Often the programmer mistakenly assumes that the program will never be used again; five years later the spaghetti code is still in use. While the tasks are accomplished and the results are accurate, the program may not be as efficient as possible and subsequent submissions may require tedious and time-consuming input and modifications.

This presentation looks at typical SAS Code, and then suggests changes to improve the efficiency and maintenance of the programs. If you are a programmer who has inherited code that was written “*many SAS versions ago*”, you will benefit from examples of “*updating your code to the current decade*”.

This tutorial focuses on maintenance - free, efficient coding techniques so that you can spend your work time being more productive!

Topics include:

- macro coding techniques
- efficient programming tips
- code reduction tricks
- maintenance-free programming suggestions.

### **INTRODUCTION**

This paper looks at example programs *written in haste*, and suggests changes to improve the efficiency and ongoing usage of the programs.

The authors acknowledge that programmers need to strike a balance to:

- minimize intervention with production jobs
- minimize runtime
- focus programming time on reusable code (not one-time queries).

### **REMOVE INEFFICIENT AND “WORDY” CODING**

Many SAS programs contain “wordy” coding that can be reduced as you learn more about the SAS language. The wordiness often results in inefficiency as data are processed multiple times, unnecessarily.

This paper provides better solutions for coding examples that:

- use multiple steps when only one step is needed
- use multiple programs when only one is needed
- take forever to run.

### **TYPICAL EXAMPLE**

Suppose your task is to generate a report that requires you to subset a SAS data set and then sort the resulting data. A common solution to this task is to use a DATA step to subset the data followed by a PROC SORT step to reorder the subset.

```
data compare;
  set report;
  if ProductCode in ('XER', 'REF', 'CRS') and Date gt '01MAY2007'd;
  keep ProductCode Usage Date Location;
run;

proc sort data=compare;
  by ProductCode Date;
run;
```

## PROBLEMS

- All observations in the REPORT data set are read but only specific ProductCode and Date values are required
- The data are read twice (by SET statement and the PROC) which could be very resource intensive if you work with lots of data.

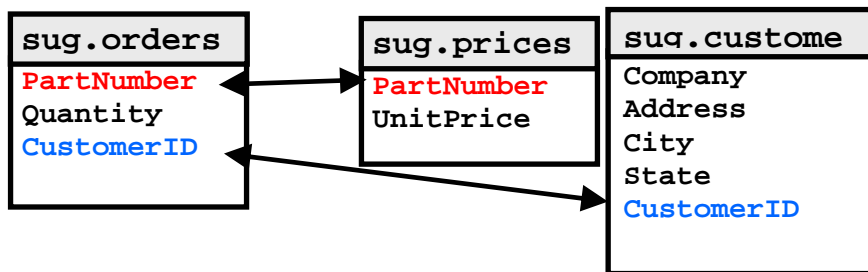
## IMPROVED EXAMPLE

An alternate solution is to subset the data as you reorder it in the PROC SORT step. In this solution, you use a WHERE statement in the PROC SORT step to subset the data, and a KEEP= data set option to choose the columns of interest. The OUT= option enables you to preserve the original table, and create a new table with the reduced data of interest.

```
proc sort data=report
  (keep=ProductCode Usage Date Location)
  out=compare
  ;
  where ProductCode in ('XER','REF','CRS') and Date gt '01MAY2007'd ;
  by ProductCode Date;
run;
```

## NEXT TYPICAL EXAMPLE – WORKING WITH MULTIPLE DATA SOURCES – NO COMMON KEYS

Your data sources often do not have common join fields. If you are a “DATA step purist” then your program would require multiple steps to join the data and get the results. Consider the following invoicing example where the data required to generate invoices is in three SAS data sources that you need to join to get produce invoices :



```
proc sort data=sug.Orders out=Orders;
  by PartNumber;
data step1;
  merge Orders(in=inOrders) sug.Prices;
  by PartNumber;
  if inOrders;
proc sort data=step1;
  by CustomerID;
proc sort data=sug.Customer out=Customer;
  by CustomerID;
data step2;
  merge step1(in=inStep1) Customer;
  by CustomerID;
  if inStep1;
  InvoiceAmt=Quantity*UnitPrice;
proc sort data=step2;
  by Company;
proc print data=step2;
  var CustomerID Company Address City State
  PartNumber Quantity InvoiceAmt;
run;
```

## PROBLEMS

Not only is the coding wordy to write but the data are read and sorted multiple times. In the real world, this could be resource-intensive.

### IMPROVED EXAMPLE

Perhaps you have not considered the advantages of SQL?! There are times that SQL can provide an easier solution, especially since common join variables are not required. This example joins 3 tables in one step, without a sort requirement. Obviously, with large data files, indexes can improve the performance of the join.

```
proc sql;
  select a.CustomerID, Company, Address,
         State, b.PartNumber, Quantity,
         Quantity*UnitPrice as InvoiceAmt
  from sug.Customers as a,
       sug.Prices as b,
       sug.Orders as c
  where a.CustomerID=c.CustomerID and
        b.PartNumber=c.PartNumber
  order by Company ;
quit;
```

### REDUCE PROGRAMMER INTERVENTION

When programs are written too specifically to the task, user-intervention is required on subsequent runs to:

- change dates and timeframes
- change titles and inclusion criteria.

Sometimes these changes require scanning 1000's of lines of code. We know – we have inherited that task on numerous occasions ☹.

Macro variables, SAS functions, and other coding tricks can be used to generalize a program making it more “data-driven”. For example,

- macro variables enable you to define values for parameters used throughout a program
- functions provide system date and time information for use in labeling, subsetting, etc.
- Macros provide decision tools that can determine which programming segments to execute based on specified criteria.

### TYPICAL EXAMPLE

Suppose your task is to run a daily report that compares month-to-date revenue for the current month to revenue for the previous month. In the solution below, the values for the current month and year, and the values for month and year for the previous month are hard coded in the program. Also, three DATA steps are used to create the desired data set for the report. The first DATA step selects the data for the current month, the second DATA step selects the data for the previous month, and the third DATA step concatenates the two data sets created by the first two DATA steps.

```
data currentmonth;
  set sug.sales;
  if month(Date) = 5 and year(Date) = 2007;
  MonthYear = " 5/2007";
run;

data lastmonth;
  set sug.sales;
  if month(Date) = 4 and year(Date) = 2007;
  MonthYear = " 4/2007";
run;

data comparemonths;
  set lastmonth currentmonth;
run;

proc means data=comparemonths;
  class MonthYear;
  var Revenue;
  title "MTD Revenue vs Last Month";
run;
```

## PROBLEMS

The solution above requires you to change the code each month and is inefficient because three DATA steps are used when only one DATA step is needed. This means that the data are read multiple times, which is resource intensive if you are processing a lot of data. An alternate solution is to use DATA step functions to eliminate the need to alter the code each month, and use the OUTPUT statement to build the final data set in one DATA step.

## BETTER EXAMPLE

```
data comparemonths;
  set sug.sales;
  if month(date)=month(today()) and year(date)=year(today()) then do;
    MonthYear= put(today() , mmyyS7.);
    output;
  end;
  else do;
    lastmonth=intnx('MONTH',today(), -1);
    if month(date)=month(lastmonth)and year(date)=year(lastmonth) then do;
      MonthYear=put(lastmonth,mmyyS7.);
      output;
    end;
  end;
run;
proc means data=comparemonths;
  class MonthYear;
  var Revenue;
  title "MTD Revenue vs Last Month";
run;
```

## TIPS

The **S** in the `mmyyS7.` format requests a SLASH to separate the month and year. If you are not familiar with this specification, have a look at SAS online documentation! Many date formats enable specification of delimiters.

The PUT function enables easy conversion from the numeric date to the desired character representation.

## EVEN BETTER EXAMPLE

If the job ran across midnight, the `today()` function would return multiple dates as the program progressed. To avoid issues with accuracy and to reduce function calls, create a macro variable at the top of the program to determine today's date (ie: when the program BEGAN execution). Then, one time define the constants that will be used while processing the data.

```
%let DateToday = %sysfunc(today());    ** Date at time program starts;

data comparemonths;
  set sug.sales;
  *** create static values - first time through the DATA step;
  retain lastmonth mon_today yr_today MonthYear_today
          mon_lastmonth yr_lastmonth MonthYear_lastmonth
  ;
  if _n_ = 1 then do;
    *** current month;
    mon_today = month(&DateToday);
    yr_today = year(&DateToday);
    MonthYear_today = put(&DateToday , mmyyS7.);
    *** last month;
    lastmonth=intnx('MONTH',&DateToday, -1);
    mon_lastmonth = month(lastmonth);
    yr_lastmonth = year(lastmonth);
    MonthYear_lastmonth =put(lastmonth,mmyyS7.);
  end;

  *** Process Current Month Revenue;
  if month(date)= mon_today and year(date)= yr_today then do;
    MonthYear= MonthYear_today;
```

```

        output;
    end;
    *** Process Last Month Revenue;
    else do;
        if month(date)= mon_lastmonth and year(date)= yr_lastmonth then do;
            MonthYear = MonthYear_lastmonth;
            output;
        end;
    end;
end;
run;
proc means data=comparemonths;
class MonthYear;
var Revenue;
title "MTD Revenue vs Last Month";
run;

```

### CONSIDER TWO FINAL IMPROVEMENTS

1. Since there may be an enormous amount of data, a WHERE clause on the SET statement would reduce the amount of data read.
2. It is likely that the 7 variables created at the top of the DATA step would be needed / used elsewhere in the program. All of these values could be created at the top of the program with %SYSFUNC and stored in macro variables.

### THINK "PRODUCTION"

If you work in an environment where you'd like to write the code once and then apply it to a broad spectrum of business problems, you will love this!

Suppose you use the same logic and reporting formats on a regular basis for each new product / campaign / region. The programs are the same except for changes to:

- Dates
- Inclusion / exclusion criteria
- Title and footnote text
- Search logic
- Output locations
- Etc.

Your current approach is to hunt (everywhere!) for the most recent version(s) you can find of the query / reporting program and then after you FINALLY find the file, you copy the code and change the name. Next you have to:

- Step through the code to locate all of the required changes
- Start typing to input the new information
- Run the code and OOPS – fix the semicolons and quotes you inadvertently overtyped
- Try again.....

### APPROACH

- Store a single dated copy of each **module** (segment of source code)
- Create a **driver** program that provides macro variable input for the current scenario and then calls the appropriate modules.

## EXAMPLE

When you are ready to start reporting for a new initiative, just provide the parameter values as input to your standard programs.

```
*** Driver Program - specify appropriate values for parameters;
%let prestart = 01OCT2006;    *** Pre campaign period for comparison;
%let prestop  = 31DEC2006;    *** End of Pre campaign period;
%let poststart = 01JAN2007;   *** start campaign tracking;
%let poststop  = 30JUN2007;   *** expire date - stop tracking;

%let title = "January 2007 Introduction of Low Interest Rates";

%let products = ('VRS', 'GQL', 'BGO', 'DLA');
%let codes = ('015', '119', '214');

*** run standard reporting programs;
%include 'CampaignReportingExtract_2007_01_10.sas';
%include 'CampaignReportingOutput_2007_01_13.sas';
```

## IMPROVED EXAMPLE

The above approach works great, until your source code changes. Then, you need to find all drivers that are calling the source, and you need to change the date stamp in the %include. Not fun! To avoid changing the **version** in 100's of drivers, use a **placeholder reference** in your drivers so that the most current source is always called.

Create a program that references the latest source version:

### Contents of CampaignReportingExtract\_CurrentPgm.sas

```
*** call latest version of source program;
%include 'CampaignReportingExtract_2007_01_10.sas';
```

### Revised contents of driver program

```
*** Driver Program - specify appropriate values for parameters;
%let prestart = 01OCT2006;    *** Pre campaign period for comparison;
%let prestop  = 31DEC2006;    *** End of Pre campaign period;
%let poststart = 01JAN2007;   *** start campaign tracking;
%let poststop  = 30JUN2007;   *** expire date - stop tracking;

%let title = "January 2007 Introduction of Low Interest Rates";

%let products = ('VRS', 'GQL', 'BGO', 'DLA');
%let codes = ('015', '119', '214');

*** run standard reporting programs;
%include 'CampaignReportingExtract_CurrentPgm.sas';
%include 'CampaignReportingOutput_CurrentPgm.sas';
```

## TESTING TIPS

When you improve a program using some of the techniques suggested above, it is prudent to compare the results to insure they are consistent and correct. Scanning output for similarity is often the approach taken, but it is easy to overlook discrepancies. Testing the equality of intermediate and final tables from the two solutions is a more accurate approach. PROC COMPARE enables comparison of entire tables, columns, or subsets of data. While many options are available, a very simple comparison of two tables is accomplished using:

```
proc compare data=sales compare=sales2;
run;
```

## **CONCLUSION**

As you learn more about SAS, you will find that there are numerous features that enable you to accomplish tasks easily and efficiently. The simple techniques provided in this paper should help you identify and correct a few inefficiencies and wordiness in your coding. As shown in the first two examples above, a common but inefficient practice used by many SAS users is to process data more times than is necessary. Additional examples showing code reduction techniques are provided in the one hour presentation.

## **CONTACT INFORMATION**

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

Marje Fecht

Email: [marje.fecht@prowerk.com](mailto:marje.fecht@prowerk.com)

Larry Stewart

Email: [larry.stewart@sas.com](mailto:larry.stewart@sas.com)

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.