Paper 043-2012

It's Now Your Project—Clean It Up and Make It Shine

Leonard Polak, Wells Fargo Technology and Operations, Irvine, CA, USA

ABSTRACT

Inherited projects may not be well designed. Though a new project owner may not have time to upgrade substandard code immediately, most inherited projects offer plenty of room for improvement. Developing department or company standards and best practices, and then applying these standards and best practices can result in code that is easy to understand, easy to maintain, and easy to transition. The examples in this paper are meant to promote improvements in productivity, consistency, and clarity.

INTRODUCTION

To paraphrase P.J. O'Rourke,[1] who was paraphrasing Milton Friedman, there are four types of projects:

1. Projects you develop for yourself.
2. Projects you develop for other people.
3. Projects other people develop for you.
4. Projects other people develop for other people.

And these four types of projects yield four results:

1. Excellent work, including pleasing design, efficient code, and thorough documentation.
2. Good work, but possibly a step down from project type one.
3. Fair work, likely a good step down from project type two.
4. Chernobyl.

Inherited projects may not be well designed. The code may be difficult to read or understand and may contain errors or omissions. Developing department or company standards and best practices, and then applying these standards and best practices can result in code that is easy to understand, easy to maintain, and easy to transition. Having easy to understand code is useful, when the purpose or benefit is requested; having easy to maintain code is useful, when the code needs to be updated; and having easy to transition code is useful, when the project owner is unavailable.

EXAMPLES

This paper presents actual inherited code, revised to run against SAS® provided sample data sets. Programs in the body of this paper can be copied and run as intended without modifications. (The code was tested using Enterprise Guide 4.3 under AIX 64 and PC SAS for Windows under XP Professional Version 2002, Service Pack 3.)

Inherited code appears in the left column and improved code appears the right column. Pre-requisite temporary tables are needed for some examples. SAS keywords appear in upper case, while user supplied text (e.g., librefs, variable names, and data values) may appear in upper case, lower case, or mixed case.

Examples briefly cover the following topics:

- Documentation
- Appearance
- Naming Objects
- Program Dates
- Unnecessary Variables
- Unnecessary Steps
- Data Set Options
- Table Lookup
- Combining Data Vertically
- Renaming Variables

DOCUMENTATION

Rarely is adequate project documentation provided with the code. If available, original project documentation can be useful in verifying that inherited code meets specifications. Having the name and location of all program files is necessary, as SAS programs can call other SAS programs. Supporting documentation explaining important decisions should be provided. Of course, security rights to all servers, files, and folders needs to be provided. Here is a bullet list of desired program documentation:

- Original project specifications, including purpose, requirements, and scope.
- The name and location of all input files.
- The name and location of all program files. A node or code window in SAS Enterprise Guide should be considered a program file.
- The name and location of all output files.
- Documentation describing non-standard and complex code.

Documentation regarding job submission would also be helpful. The new project owner should know the order in which programs, nodes, or steps are to be submitted. And to know if jobs are submitted based on date and time, a control file, or ad hoc. Here is a bullet list of desired job documentation:

- A task list sorted in order of performance.
- Information regarding when the job should be submitted.
- A data customer list.

Program 1A shows a common occurrence, no documentation provided. Program 1B shows sample documentation.[2]

```
/*Program 1A*/
```

```
/*Program 1B
Program         Your_Project.sas
Location        C:\Documents and Settings\Leonard Polak\My
                Documents\My SAS Files
Description     Develop sample code for presentation
Author          Leonard Polak
Created         20110809
Status          Complete
Validation      Reviewed log.
Input           sashelp.class
                sashelp.prdsal2
                sashelp.prdsal3
                sashelp.zipcode
Output          Temporary data sets only.
Notes           This program is complete; there is no called code.
                Documentation regarding non-standard or complex
                code is included in Revision History or with the
                code.
                No task list is needed; job may be submitted as
                desired.
                No customer list needed.
                Security rights to all servers, files, and
                folders have been verified.

_____________________________________________________________________

Revision history
#     Date          Author           Description
01    20110810      Leonard Polak    Changed variable name case to
                                     upper for prdsal data sets, to
                                     match convention.*/
```

APPEARANCE

Inherited code may not have a pleasing look. Maintaining code that is easy to read is a best practice.

All code in programs 2a and 2c are in lower case. No lines are indented and no blank line separates steps. And 2c shows a RUN statement, rather than a QUIT statement. Programs 2B and 2D show standard capitalization and indented lines; also one line per statement and a blank line between steps.

```
/*program 2a*/
proc sort data = sashelp.class
out=tbl2;
by sex age;run;
data tbl3;set tbl2;
by sex age;
if first.sex;run;
```

```
/*Program 2B*/
PROC SORT DATA = sashelp.class
          OUT  = sortClass;
  BY Sex Age;
RUN;

DATA firstSortClass;
  SET sortClass;
  BY Sex Age;
  IF FIRST.Sex;
RUN;
```

```
/*program 2c*/
proc sql;
create table oldCount as select sex, count(sex)
as count
from sashelp.class
group by sex
having count>9
order by sex;
run;
```

```
/*Program 2D*/
PROC SQL;
  CREATE TABLE newCount AS
    SELECT Sex, COUNT(Sex) as Count
    FROM sashelp.class
    GROUP BY Sex
    HAVING count > 9
    ORDER BY Sex;
QUIT;
```

NAMING OBJECTS

Objects may be named without thought. A data set name of temp1 or a variable name of var1 generally conveys no meaning. Choosing descriptive librefs, data set names, and variable names is best.

Program 3A shows the coder's initials as the libref. Later in the process, the coder sends current period output to the library AP and moves older data sets to an annual archive library, AP2. In this case, having multiple folders for the same periodic output is unnecessary. So are the RUN statements. Program 3B shows a single LIBNAME statement with a libref of outLib, which stands for output library. Program 3C shows a libref of X, which conveys no information regarding the following folder. Further, the X libref may be confused with the X command. Like program 3B, program 3D shows a libref of outLib.

```
/*Program 3A*/
LIBNAME AP '/project_name';RUN;
/*LIBNAME AP2
'/project_name/archive/2010';RUN;*/
LIBNAME AP2 '/project_name/archive/2011';RUN;
```

```
/*Program 3B*/
LIBNAME outLib '/project_name/output';
```

```
/*Program 3C*/
LIBNAME X 'C:\Documents and Settings\Leonard
Polak\My Documents\My SAS Files';RUN;
```

```
/*Program 3D*/
LIBNAME outLib 'C:\Documents and
Settings\Leonard Polak\My Documents\My SAS
Files';
```

PROGRAM DATES

Hard coded date constants force the coder to routinely review the complete process and make manual updates. A better option is using macro date variables. Adding a run date, which (by convention) is the last date in the report period, is a best practice. Additional macro variable dates are based on the run date.

In program 4A, the date constants need to be reviewed and changed, as needed. (Technically, 'date' is a date variable and 'year' is a numeric variable.) In program 4B, the hard coded date literals are replaced with macro date variables. By placing the macro variable values for interval and runDate at the top of the program, the coder does not need to review the complete process for dates to change. (As 'DATE' is a system option and format and 'YEAR' is function and format, these variable names are not good choices.) The name YCODE_YEAR stands for Year, Current, 0 (offset number), Day, and End—and YEAR (format). Additional information on the naming convention and more macro date variable examples can be found in the appendix.[3]

```
/*Program 4A*/
DATA dateConstant1;
  SET sashelp.prdsal2
   (RENAME = (monyr = DATE));
  IF date >= '01JAN1998'd;
  IF date <= '31DEC1998'd;
RUN;

DATA dateConstant2;
  SET sashelp.prdsal3;
  IF year = 1998;
RUN;
```

```
/*Program 4B*/
%LET interval   = -13;
%LET runDate    = %SYSFUNC(TODAY());
%LET runDate    = %SYSFUNC(INTNX(YEAR,&runDate,&interval,END));
%LET YC0DB      = %SYSFUNC(INTNX(YEAR,&runDate,0,BEGIN));
%LET YC0DE_YEAR = %SYSFUNC(PUTN(&runDate,YEAR.));

%PUT %SYSFUNC(PUTN(&runDate,MMDDYY10.));
%PUT %SYSFUNC(PUTN(&YC0DB,MMDDYY10.));
%PUT &YC0DE_YEAR;

DATA dateMacroVariable1;
  SET sashelp.prdsal2
   (WHERE = (monyr >= &YC0DB and monyr <= &runDate));
RUN;

DATA dateMacroVariable2;
  SET sashelp.prdsal3 (WHERE = (year = &YC0DE_YEAR));
RUN;
```

UNNECESSARY VARIABLES

New variables are easy to create, but may not be needed. Minimizing the number of variables can improve performance and readability.

Program 5A shows a date value being changed from the first day of the month to the last day of the month (The abbreviation eom stands for end of month.) This coder creates three DATA steps to get the new date value, and note the missing QUIT statement. The DATA step in program 5B is an improvement over the program in 5A. The PROC SQL procedure in program 5B achieves the same result as the preceding DATA step without an assignment statement, drop statement, or rename statement.

```
/*Program 5A*/
DATA dateUpdate;
  SET sashelp.prdsal3;
  eomDate = INTNX('MONTH',date,0,'END');
  FORMAT eomDate MMDDYY10.;
  DROP date;
RUN;

DATA oldEom;
  SET dateUpdate;
  RENAME eomDate = DATE;
RUN;

PROC DATASETS;
  DELETE dateUpdate;
RUN;
```

```
/*Program 5B*/
DATA newEom (DROP = date
           RENAME = (eomDate = DATE));
  SET sashelp.prdsal3;
  eomDate = INTNX('MONTH',date,0,'END');
  FORMAT eomDate MMDDYY10.;
RUN;

PROC SQL;
  CREATE TABLE newDate AS
    SELECT
      country,
      state,
      county,
      actual,
      predict,
      prodtype,
      product,
      year,
      quarter,
      month,
      INTNX('MONTH',date,0,'END')
        FORMAT = MMDDYY10. as DATE
    FROM sashelp.prdsal3;
QUIT;
```

UNNECESSARY STEPS

Creating unnecessary DATA steps or PROC steps is routinely done, though not a best practice. Producing extra steps may decrease performance and readability.

Program 6A shows a DATA step created solely to add a calculated variable, then a second DATA step created to exclude records based on the preceding results. As shown in program 6B, the same results can be achieved with one DATA step. Program 6C shows two PROC SQL steps. (The two DATA steps in 6C were added to create different values for the height and weight variables, as the class data set is being used three times.) The coder has chosen to

add a second PROC SQL, rather than integrate the class3 data set. Program 6D returns the same results as 6C, with less coding.

```
/*Program 6A*/
DATA addFlag;
  SET sashelp.prdsal3;
  IF actual > predict THEN flag = 1;
RUN;

DATA oldExclude (DROP = flag);
  SET addFlag;
  WHERE flag = 1;
RUN;
```

```
/*Program 6B*/
DATA newExclude;
  SET sashelp.prdsal3 (WHERE = (actual > predict));
RUN;
```

```
/*Program 6C*/
DATA class2;
  SET sashelp.class;
  Height = Height - 10;
RUN;

DATA class3;
  SET sashelp.class;
  Weight = Weight - 10;
RUN;

PROC SQL;
  CREATE TABLE joinClass AS
    SELECT
      class.Name,
      class.Sex,
      class.Age,
      class2.Height
    FROM
      sashelp.class
        INNER JOIN class2
          ON class.Name = class2.name;
QUIT;

PROC SQL;
  CREATE TABLE joinClass2 AS
    SELECT
      joinClass.Name,
      joinClass.Sex,
      joinClass.Age,
      joinClass.Height,
      class3.Weight
    FROM
      joinClass
        INNER JOIN class3
          ON joinClass.Name =
              class3.Name;
QUIT;
```

```
/*Program 6D*/
DATA class2;
  SET sashelp.class;
  Height = Height - 10;
RUN;

DATA class3;
  SET sashelp.class;
  Weight = Weight - 10;
RUN;

PROC SQL;
  CREATE TABLE newJoinClass AS
    select
      class.Name,
      class.Sex,
      class.Age,
      class2.Height,
      class3.Weight
    FROM
      sashelp.class
        INNER JOIN class2
          ON class.Name = class2.Name
        INNER JOIN class3
          ON class.Name = class3.Name;
QUIT;
```

DATA SET OPTIONS

Coders may use statement options, rather than data set options. (SAS offers three types of options: system options, statement options, and data set options.) Data set options often prove more efficient. The difference is context, when the option is applied.

Program 7A shows three statement options and program 7B shows three data set options. Program 7B provides better performance—as records and variables are excluded earlier in the process.

```
/*Program 7A*/
DATA statementOption;
  SET sashelp.prdsal2;
  WHERE year = 1998;
  RENAME monyr = DATE;
  KEEP monyr product actual;
RUN;
```

```
/*Program 7B*/
DATA dataSetOption (DROP = year);
  SET sashelp.prdsal2
   (WHERE  = (year  = 1998)
    RENAME = (monyr = DATE)
    KEEP   =  year monyr product actual);
RUN;
```

TABLE LOOKUP

Programs can be littered with hard to follow IF-THEN statements and WHERE statements. The coder may be able to avoid these statements by using the PROC FORMAT procedure. With PROC FORMAT, the coder may develop simpler code with fewer variables.

Program 8A shows a long IF-THEN/ELSE statement and program 8B shows PROC FORMAT, followed by a short DATA step. Both programs return the same result. Note that the format could be added to a catalog and made available to other programs. With one row per value, the format appears easier to maintain. And the values can be easily copied to Microsoft Excel and sorted by region code, rather than state. Format source: http://en.wikipedia.org/wiki/List_of_regions_of_the_United_States, see Standard Federal Regions.

```
/*Program 8A*/
DATA oldRegion;
  LENGTH Region $4;
  SET sashelp.zipcode;
  IF statename IN ('Connecticut',
'Maine', 'Massachusetts', 'New Hampshire',
'Rhode Island', 'Vermont') THEN Region = 'I';
      ELSE IF statename IN ('New Jersey', 'New
York', 'Puerto Rico', 'Virgin Islands') THEN
Region = 'II';
      ELSE IF statename IN ('Delaware',
'District of Columbia', 'Maryland',
'Pennsylvania', 'Virginia', 'West Virginia')
THEN Region = 'III';
      ELSE IF statename IN ('Alabama',
'Florida', 'Georgia', 'Kentucky',
'Mississippi', 'North Carolina', 'South
Carolina', 'Tennessee') THEN Region = 'IV';
      ELSE IF statename IN ('Illinois',
'Indiana', 'Michigan', 'Minnesota', 'Ohio',
'Wisconsin') THEN Region = 'V';
      ELSE IF statename IN ('Arkansas',
'Louisiana', 'New Mexico', 'Oklahoma', 'Texas')
THEN Region = 'VI';
      ELSE IF statename IN ('Iowa', 'Kansas',
'Missouri', 'Nebraska') THEN Region = 'VII';
      ELSE IF statename IN ('Colorado',
'Montana', 'North Dakota', 'South Dakota',
'Utah', 'Wyoming') THEN Region = 'VIII';
      ELSE IF statename IN ('Arizona',
'California', 'Hawaii', 'Nevada', 'American
Samoa', 'Guam', 'Northern Mariana Islands',
'Trust Territory of the Pacific Islands') THEN
Region = 'IX';
      ELSE IF statename IN ('Alaska', 'Idaho',
'Oregon', 'Washington') THEN Region = 'X';
RUN;

PROC FREQ DATA = oldRegion;
  TABLES Region;
RUN;
```

```
/*Program 8B*/
PROC FORMAT;
  VALUE $ fmtRegion
    'Alabama'                  = 'IV'
    'Alaska'                   = 'X'
    'American Samoa'           = 'IX'
    'Arizona'                  = 'IX'
    'Arkansas'                 = 'VI'
    'California'               = 'IX'
    'Colorado'                 = 'VIII'
    'Connecticut'              = 'I'
    'Delaware'                 = 'III'
    'District of Columbia'     = 'III'
    'Florida'                  = 'IV'
    'Georgia'                  = 'IV'
    'Guam'                     = 'IX'
    'Hawaii'                   = 'IX'
    'Idaho'                    = 'X'
    'Illinois'                 = 'V'
    'Indiana'                  = 'V'
    'Iowa'                     = 'VII'
    'Kansas'                   = 'VII'
    'Kentucky'                 = 'IV'
    'Louisiana'                = 'VI'
    'Maine'                    = 'I'
    'Maryland'                 = 'III'
    'Massachusetts'            = 'I'
    'Michigan'                 = 'V'
    'Minnesota'                = 'V'
    'Mississippi'              = 'IV'
    'Missouri'                 = 'VII'
    'Montana'                  = 'VIII'
    'Nebraska'                 = 'VII'
    'Nevada'                   = 'IX'
    'New Hampshire'            = 'I'
    'New Jersey'               = 'II'
    'New Mexico'               = 'VI'
    'New York'                 = 'II'
    'North Carolina'           = 'IV'
    'North Dakota'             = 'VIII'
    'Northern Mariana Islands' = 'IX'
    'Ohio'                     = 'V'
    'Oklahoma'                 = 'VI'
    'Oregon'                   = 'X'
    'Pennsylvania'             = 'III'
    'Puerto Rico'              = 'II'
    'Rhode Island'             = 'I'
```

```
    'South Carolina'           = 'IV'
    'South Dakota'             = 'VIII'
    'Tennessee'                = 'IV'
    'Texas'                    = 'VI'
    'Trust Territory of the Pacific Islands'
                               = 'IX'
    'Utah'                     = 'VIII'
    'Vermont'                  = 'I'
    'Virgin Islands'           = 'II'
    'Virginia'                 = 'III'
    'Washington'               = 'X'
    'West Virginia'            = 'III'
    'Wisconsin'                = 'V'
    'Wyoming'                  = 'VIII'
     other                     = 'Missing';
RUN;

DATA newRegion;
  SET sashelp.zipcode;
  Region = statename;
  FORMAT Region fmtRegion.;
RUN;
```

COMBINING DATA VERTICALLY

Combining data sets, adding a new data set to an existing one, is a common task. But this practice can result in problems. For one, these combined data sets can become quite large. But more important, these large data sets can be difficult to rebuild, should an error or problem occur. Rather than one larger data set, smaller data sets may be kept by some grouping, e.g., month or year, and then combined as needed.

Programs 9A and 9C show two methods of combining tables. Program 9A shows two tables combined with a SET statement and program 9C shows a data table added to a base table using the PROC APPEND procedure. (The PROC DATASETS procedure is included to avoid writing over the permanent data sets.) Program 9B shows a macro that splits the source data into yearly files. This is a one-time action, as future data sets contain yearly data. Program 9D shows a data set list, note the use of the colon. The data set lists feature became available with SAS 9.2, see http://support.sas.com/kb/35/419.html.

```
/*Program 9A*/
DATA oldVertical;
  SET sashelp.prdsal2 (RENAME = (monyr = date))
      sashelp.prdsal3;
RUN;
```

```
/*Program 9B*/
%MACRO splitFile(year);
  DATA prdsal_&year;
    SET sashelp.prdsal2 (RENAME = (monyr =
                                   date))
        sashelp.prdsal3;
    IF year = &year THEN OUTPUT prdsal_&year;
  RUN;
%MEND splitFile;

%splitFile(1995)
%splitFile(1996)
%splitFile(1997)
%splitFile(1998)
```

```
/*Program 9C*/
PROC DATASETS NOLIST;
  COPY IN = sashelp OUT = work;
    SELECT prdsal2 prdsal3;
QUIT;
RUN;

PROC APPEND
  BASE = prdsal2
  DATA = prdsal3 (RENAME = (date = monyr));
RUN;
```

```
/*Program 9D*/
DATA newVertical;
  SET prdsal_:;
RUN;
```

RENAMING VARIABLES

Some coders twist themselves in circles with the repeated renaming of variables. Employing a standard, e.g., keep the original variables names throughout a project, can make code easier to work with and understand.

Program 10A shows a DATA step created to rename a variable, the renamed variable used in a PROC FREQ procedure, and then renamed back to the original in another DATA step. Program 10B achieves the same result in one step, with no renaming of the variable in the source data set. Program 10C shows PROC SQL with two variables renamed. Program 10D is included to demonstrate the standard: use the given variable name during the process, only renaming variables near the end of the process.

```
/*Program 10A*/
DATA oldRename;
  SET sashelp.class;
  RENAME Sex = Gender;
RUN;

PROC FREQ DATA = oldRename;
  TABLES Gender / OUT = oldFreq;
RUN;

DATA oldFinal;
  SET oldRename;
  RENAME Gender = Sex;
RUN;
```

```
/*Program 10B*/
PROC FREQ DATA = sashelp.class
       (RENAME = (Sex = Gender));
  TABLES Gender / OUT = newFreq;
RUN;

PROC PRINT DATA = sashelp.class;
RUN;
```

```
/*Program 10C*/
PROC SQL;
  CREATE TABLE rename AS
    SELECT
      year,
      prodtype,
      product,
      SUM(actual) FORMAT = DOLLAR10. AS
        SALES,
      SUM(predict) FORMAT = DOLLAR10. AS
        ESTIMATED
    FROM
      sashelp.prdsal3
    GROUP BY
      year,
      prodtype,
      product
    ORDER BY
      year,
      prodtype,
      product;
QUIT;
```

```
/*Program 10D*/
PROC SQL;
  CREATE TABLE laterRename AS
    SELECT
      year,
      prodtype,
      product,
      SUM(actual) FORMAT = DOLLAR10. AS
        actual,
      SUM(predict) FORMAT = DOLLAR10. AS
        predict
    FROM
      sashelp.prdsal3
    GROUP BY
       year,
       prodtype,
       product
    ORDER BY
       year,
       prodtype,
       product;
QUIT;

PROC DATASETS LIB = work NOLIST;
  MODIFY laterRename;
    RENAME actual  = SALES
           predict = ESTIMATED;
QUIT;
RUN;

PROC PRINT DATA = laterRename;
RUN;
```

CONCLUSION

Though a new project owner may not have time to upgrade substandard code immediately, most inherited projects offer plenty of room for improvement. The examples in this paper are meant to promote improvements in productivity, consistency, and clarity. Many more examples are possible, and different standards could be chosen. Again, develop department or company standards and best practices and apply them.

Topic	Recommendation
Documentation	Maintain accurate, useful, and thorough documentation.
Appearance	Make code easy to read: indent; capitalize; one line per statement; add blank lines.
Naming Objects	Provide meaningful object names.
Program Dates	Add macro date variables.
Unnecessary Variables	Do not add unnecessary variables.
Unnecessary Steps	Do not add unnecessary steps.
Data Set Options	Use data set options and statement options appropriately.
Table Lookup	Use PROC FORMAT.
Combining Data Vertically	Use data set lists to combine smaller data sets.
Renaming Variables	Rename variables at the end of a project, if required.

Table 1. Standards and Best Practices

END NOTES

[1] "The Friedmans [Milton and Rose Friedman in their book, *Free to Choose*] argued that there are only four ways to spend money:

- Spend your money on yourself.
- Spend your money on other people.
- Spend other people's money on yourself.
- Spend other people's money on other people.

If you spend your money on yourself, you look for the best value at the best price.... If you spend money on other people, you still worry about price, but you may not know—or care—what the other people want.... If you spend other people's money on yourself.... And if you spend other people's money on other people, any **** thing will do and the **** with what is costs. Almost all government spending falls into category four. This is how the grateful residents of Ukraine got Chernobyl."

[2] See also Hord and Zhou (2002) for a macro to build header documentation.

[3] The following table is adapted from Automating Report Dates and Formats Using SAS®9 Software.

Position	1	2	3	4	5	6	7
Description	Month or Year	Past, Current, or Future	Offset Number	Day	Begin, Middle, End, Same	Delimiter	Format
Abbreviation	M, Y	P, C, F		D	B, M, E, S		
Example	M	P	1	D	E	_	YYMMDDN8

Table 2. Macro Date Variable Nomenclature

```
/*Date Macro Variable Examples*/
%LET increment       = -1;
%LET runDate         = %SYSFUNC(TODAY());
%LET runDate         = %SYSFUNC(INTNX(MONTH,&runDate,&increment,END));
%LET MP1DE_YYMMDDN8  = %SYSFUNC(PUTN(%SYSFUNC(INTNX(MONTH,&runDate,-1,END)),YYMMDDN8.));
%LET MC0DS_YYMMDDN8Q = %SYSFUNC(QUOTE(%SYSFUNC(PUTN(&runDate,YYMMDDN8.))));
%LET MC0DS_YYMMN6    = %SYSFUNC(PUTN(&runDate,YYMMN6.));
%LET MC0DS_YYQ       = %SYSFUNC(PUTN(&runDate,YYQ.));
%LET YP1DB_YYMMDDN8  = %SYSFUNC(PUTN(%SYSFUNC(INTNX(YEAR,&runDate,-1,BEGIN)),YYMMDDN8.));
%LET YP1DE_YYMMDDN8  = %SYSFUNC(PUTN(%SYSFUNC(INTNX(YEAR,&runDate,-1,END)),YYMMDDN8.));
%LET QP3DS_YYQ       = %SYSFUNC(PUTN(%SYSFUNC(INTNX(QTR,&runDate,-3,END)),YYQ.));
%LET MC0D20          = %SYSFUNC(MDY(%SYSFUNC(MONTH(&runDate)),20,%SYSFUNC(YEAR(&runDate))));

%PUT %SYSFUNC(putn(&runDate,mmddyy10.));
%PUT &MP1DE_yymmddn8;
%PUT &MC0DS_yymmddn8q;
```

```
%PUT &MC0DS_yymmn6;
%PUT &MC0DS_yyq;
%PUT &YP1DB_yymmddn8;
%PUT &YP1DE_yymmddn8;
%PUT &QP3DS_yyq;
%PUT %SYSFUNC(putn(&MC0D20,mmddyy10.));
```

REFERENCES

- Hord, Chris and Zhou, Jay. "SAS® Macros to Help Relieve Common Program Documentation Pain." Available at http://www2.sas.com/proceedings/sugi27/p208-27.pdf.
- Levin, Lois. "SAS® Programming Guidelines." Available at http://www2.sas.com/proceedings/sugi31/123-31.pdf.
- O'Rourke, P. J. 1998. *Eat the Rich*. New York, NY: Atlantic Monthly Press. Available at http://books.google.com/books?id=YzTauHNa4FIC&pg=PT295&lpg=PT295&dq=%22p.j.+o%27rourke%22+and+%22chernobyl%22&source=bl&ots=Xdt4Sxkbaw&sig=cw2b6Am57DRGAsE17m6eBEPdy1s&hl=en&ei=905ATvuiBLLgsQLTmqkT&sa=X&oi=book_result&ct=result&resnum=3&ved=0CCMQ6AEwAg#v=onepage&q&f=false
- Simeoni, John and Coy, Dikki. "Automating Report Dates and Formats Using SAS®9 Software." Available at http://support.sas.com/resources/papers/proceedings10/075-2010.pdf.

ACKNOWLEDGMENTS

Thanks to Ravi Kumar for ideas in the Combining Data Vertically section.

CONTACT INFORMATION

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

Leonard Polak
Wells Fargo Technology and Operations
23 Pasteur
Irvine, California 92618
MAC E2718-020
Tel 949-753-3807
leonard.polak@wellsfargo.com