Paper 7260-2016

No More Bad Dates!: A Guide to SAS® Dates in Macro Language

Kate Burnett-Isaacs, Statistics Canada; Andrew Clapson, MD Financial Management

ABSTRACT

The SAS® macro language is an efficient and effective way to handle repetitive processing tasks. One such task is conducting the same DATA steps or procedures for different time periods, such as every month, quarter, or year. SAS® dates are based on the number of days between January 1st, 1960, and the target date value, which, while very simple for a computer, is not a human-friendly representation of dates. SAS dates and macro processing are not simple concepts themselves, so mixing the two can be particularly challenging, and it can be very easy to end up working with bad dates! Understanding how macros and SAS dates work individually and together can greatly improve the efficiency of your coding and data processing tasks. This paper covers the basics of SAS macro processing, SAS dates, and the benefits and difficulties of using SAS dates in macros, to ensure that you never have a bad date again.

INTRODUCTION

To understand how SAS dates interact with the macro language, we first must have a better understanding of SAS macro processing. SAS macro code consists of macro references, specifically macro statements, variables, calls, and definitions. Prior to the execution of any DATA step or procedure, SAS checks to see if macro references exist. If they do, SAS passes the macro reference to the macro processor which resolves the code and executes the DATA steps and procedures.

MACRO VARIABLES

What makes macro programming so efficient is the ability to use macro variables as parameters within SAS DATA step and procedure code. A macro variable can be defined in many ways, but one of the most common is by using the '%let' statement, followed by the macro variable name, an equals sign, and the text value to be assigned to the macro variable. The name of macro variables must follow the same conventions as names of data set variables, i.e. they must be between 1-32 characters long and begin with a letter or underscore (_). The syntax for a macro variable is as follows:

```
%let macroVarName = text;
```

To use (resolve) a macro variable within your code, the macro variable must referenced beginning with an ampersand '&' and endingwith a period '.':

```
&macroVarName.
```

Note: Although the closing period above is not strictly necessary, it is good coding practice.

Timing matters when resolving macro variables as they are resolved - the value 'text' replaces `&macroVarName.` in our code - prior to the execution of the code. Therefore we cannot use a macro variable in the same DATA step or procedure in which it is created.

USING MACRO VARIABLES

Unlike values in a SAS data set, macro variables are not defined as explicitly character or numeric; they are simply stored and treated as text. This is very important to keep in mind when using date values as input text for macro variables. The beauty of this type-ambiguity is that the resolved macro value can perform a variety of roles depending on where and how the macro variable is resolved. For example, we can define a macro variable to represent a data set name:

```
%let data_set = set1;
```

And then later on in our code we can use the macro variable `&data_set.` to print all observations in the data set 'set1':

```
proc print data = &data_set.;
```

```
run;
```

In this example, the macro variable reference `&data_set.` will resolve to 'set1'. In order to use this same code to print the observations for another data set, we simply redefine the macro variable and call the macro again:

```
%let data_set = set2;
proc print data = &data_set.;
run;
```

Without duplicating the code for the PRINT procedure, the above example will now print observations for the data set 'set2'.

DEFINING A MACRO

In addition to individual macro variables, we can also define a macro, which is essentially the same thing as a macro variable – a block of text that can be used and reused as necessary. To define a macro, we use the macro statement '%macro', followed by the macro name, and close the macro definition must close with '%mend'.

```
%macro macroName;
   data WORK.Sales_2;
         set WORK.Sales_1;
   run;
%mend;
```

A macro is called (invoked) by using the macro name after a percentage sign:

```
%macroName
```

It is not required to follow a macro call with a semicolon (and in some cases this will even cause errors). Note that even though a SAS macro can contain very sophisticated code (and can even contain macros that call OTHER macros); all SAS macros are based on text substitution. In the above example, `%macroName` is simply a shorthand that substitutes for the entire block of DATA step code.

USING SAS FUNCTIONS WITH MACROS

Because macro variables are simple text strings, processing or formatting their values often isn't quite as straightforward as dealing with values in a SAS data set. Similarly, DATA step functions cannot be applied to macro variables in quite the same fashion as within a DATA step. However, the macro function %SYSFUNC(or %QSYSFUNC) allows us to use a DATA step function to define a macro variable. The syntax looks like:

```
%SYSFUNC(function-name(function-argument), <format>);
```

Because of the timing issue discussed earlier, the familiar PUT() and INPUT() functions will not work with macro variables even when using the %SYSFUNC() wrapper, because they require values to be specified at compile time. Instead, we must use the INPUTN() and INPUTC() functions to specify numeric or character informats at runtime, or the PUTN() and PUTC() functions to specify numeric or character formats at run time. Here's an example that uses the PUTN() function with a macro variable:

```
%let exampleVar = %SYSFUNC(PUTN(&macrovar., format);
```

USEFUL MACRO STATEMENTS

In macro code, just as in DATA steps, we often want to use conditional logic to create variables or execute code. The macro analogue of the DATA step's IF/THEN/ELSE statements are, unsurprisingly: %IF/%THEN/%ELSE. These conditional macro logic statements do not operate on DATA step variables; instead they determine which sections of text within a macro are read by the macro processor. Though this sounds like a restriction, it is in fact the opposite: we can now use conditional logic literally anywhere in our code and are no longer limited to the inside of a DATA step.

In the same vein, iterative DATA step code can be replicated in macro form, with a '%'in front. The '%DO', '%DO %UNTIL', '%DO %WHILE' statements perform similar activities as their DATA step counterparts, but again they can be used outside of the DATA step and anywhere in a macro.

A useful tool to view what our macro variables resolve to is the %PUT statement. This statement returns the text and resolved macro variables to the log. In the above example, if we wanted to check the assigned value for the macro variable 'data_set', we would submit the following:

```
%put data_set = *&data_set.*;
```

And SAS would print the following result to the log, confirming our result:

```
data_set = *set2*
```

INTRODUCTION TO SAS DATES

SAS has three different counters to keep track of dates and time. Though this paper focuses on the date counter, the general principles of handling SAS dates applies to handling time counters as well. A SAS date value records the number of days between January 1, 1960 and the date value. If the date is prior to January 1, 1960, then the date value is negative and if the date is after January 1, 1960, the date value is positive. Keeping track of dates is always relative to a specific point in time, whether for humans or SAS, however January 1, 1960 is not a period in time we are used to reading, therefore, SAS dates rarely mean anything to us.

For example, January 1, 2015 is the SAS date value '20089'. In order to understand SAS dates, we must format them to a value that makes sense to us. Using formats, we can transform SAS dates into an understandable date value. There are many different date formats that can be used. Some of the more common date formats include DATEw., DDMMYYw., and MMDDYYw., where 'w' is the length of the date being displayed. Table 1summarizes popular format types, their descriptions and results.

Format	Description	Results
DATE9.	This is a 9 digit date value where the day of the month is followed by the three letter month abbreviation and then the year	15JAN2015
DDMMYY10.	The ten digit day/numerical month/year	15/01/2015
MMDDYY10.	The ten digit numerical month/day/year	01/15/2015
YYMMN6.	The six digit year followed by numerical month	201501
MonYY7.	The seven digit value of the three letter month abbreviation and then the year	JAN2015

Table 1. Some Example SAS Date Formats

DATE FUNCTIONS

Given how troublesome it can be expressing and understanding a date as an integer value, why bother with SAS dates at all? Because working with SAS date values enable access a variety of very useful functions, procedures, and formats that will vastly simplify your dating life. Working with SAS dates simplifies such tasks as calculating date intervals and incrementing dates by specific periods.

In order to determine the number of periods between two SAS dates we use the INTCK() function. To increment dates, we use the INTNX() function:

```
INTCK('interval', start-period, end-period)
INTNX('interval', start-period, number-of-increments, alignment)
```

Where 'interval' is day, month, quarter or year between the start and end dates, and (for INTNX()) the number of increments is the number of days, months, quarters or years to be added to the start period and the alignment is the beginning (B), middle (M) or end (E) of the interval period.

SAS DATES IN MACROS

Now that we have covered the basics of macros and SAS dates we can get to the exciting part! Recalling that macros allow for the simplification of repetitive processing, we can now extend this functionality by calling the same SAS code for every month, quarter, or year. The unique behavior of both SAS macros and SAS date values must be taken into consideration when using SAS dates as values for macro variables. As mentioned earlier in this paper, macro variable values are stored as text. Therefore, if we have a macro variable in a date format we understand for January 2015 we could write it as such:

```
%let date = 201501;
```

However, to SAS, this is a bad date! It is **not** January 2015, rather this date is 201,501 days after January 1, 1960, or in our language: September 10, 2511. Unless you've developed one HECK of a predictive model, this probably isn't the date you were aiming for.

In order to use the macro variable *date* as a SAS date, we will have to use the appropriate *informat*. To convert our macro variable into a SAS date we would use an INPUTN() function. This function specifies a numeric informat at runtime. However, we now know that we can't use any DATA step function with macro code unless we also include the %SYSFUNC wrapper. The syntax to format our macro variable date is as follows:

```
%let formatDate = %sysfunc(inputn(&date., YYMMN6.));
```

Now, the macro variable `formatDate` will appear to us as 201501 (due to the YYMMM6. format used in the INPUT()function), but SAS will recognize the value as a formatted date and store it as a SAS date value of 20089.

EXAMPLE

The best way to understand how SAS dates and macros work together is to go through an example. Let's look at a case where we are the owner of a dating service company called 'Great Dates '. At the end of every quarter we'll want to calculate our total expenses and net income for that quarter. The data set, called 'IncomeStatement', looks as follows in Figure 1:

	Date	Operation	OperationType	Value
1	201503	Selling	Expense	7000
2	201503	Admin	Expense	6000
3	201503	Sales	Sales	100000
4	201503	GdsSold	Cost	75000
5	201504	Selling	Expense	6000
6	201504	Admin	Expense	5000
7	201504	Sales	Sales	105000
8	201504	GdsSold	Cost	80000
9	201505	Selling	Expense	7500
10	201505	Admin	Expense	7000
11	201505	Sales	Sales	110000
12	201505	GdsSold	Cost	70000
13	201506	Selling	Expense	6500
14	201506	Admin	Expense	5500
15	201506	Sales	Sales	100000
16	201506	GdsSold	Cost	75000

Figure 1. Income Statement of Great Dates

Because we'll be doing this calculation every quarter, we will write a macro that calculates total expenses and net income and our macro variable is the date we want the calculation to be for.

```
%let date=201506;
%macro basicdates;
   data NetIncome_1;
         set IncomeStatement;
```

```
      where Date=&date.;
run;

proc transpose data = NetIncome_1 out = NetIncome_2;
var Value;
      by Date;
      id Operation;
run;

data NetIncome_3;
      set NetIncome_2;
            TotalExpense=sum(Selling, Admin, GdsSold);
            NetIncome=(Sales-TotalExpense);
run;

%mend;
%basicdates
```

The case above does not informat the macro variable date and so SAS resolves `&date.` in the datastepNetIncome_1 and generates the following code:

```
data NetIncome_1;
      set IncomeStatement;
where Date=201506;
run;
```

The SAS date value 201506 (which is interpreted as the integer value 201,506) does not exist in our data set and we get the following result for Output 1:

```
NOTE: There were 0 observations read from the data set
WORK.INCOMESTATEMENT.
      WHERE Date=201506;
NOTE: The data set WORK.NETINCOME_1 has 0 observations and 4 variables.
NOTE: DATA statement used (Total process time):
real time           0.01 seconds
cpu time            0.01 seconds
```

Output 1. Output from NetIncome Code

In this case, we have used a bad date! In order to get the result we want, we have to convert the macro variable date value to a SAS date. We do this by using the INPUTN() function:

```
%let date=201506;
%macro formatdates;
   %let formatdate=%sysfunc(inputn(&date., YYMMN6.));

   %put My dates is &date. and its SAS date is &formatdate.;

   data NetIncome_1;
         set IncomeStatement;
               where Date=&formatdate.;
   run;

   proc transpose data = NetIncome_1 out = NetIncome_2;
   var Value;
   by Date;
         id Operation;
   run;

   data NetIncome_3;
```

```
    set NetIncome_2;
          TotalExpense=sum(Selling, Admin, GdsSold);
          NetIncome=(Sales-TotalExpense);
    keep date TotalExpense NetIncome;
    run;

%mend;
%formatdates
```

The %PUT statement allows us to see what the macro variables date and 'formatDate' resolve to, and so what SAS is using in the WHERE statement in NetIncome_1. The %PUT statement prints the following results to the log:

```
My date is 201506 and its SAS date is 20240
```

Now we know we are using the correct date! The resulting net income and total expenses are shown in Figure 2:

	Date	TotalExpense	NetIncome
1	201506	87000	13000

Figure 2 - Correct Income and Total Expenses

Let's have a little more fun with macros and dates and expand the task to calculating total expenses and net income for the whole second quarter. There are a few approaches that we can take to tackle this task, but in order to use all the tools we have just learned, let's start by defining our date macro variable as the first month of the second quarter:

```
%let date=201504;
```

We want to include all the months of the quarter in question in our calculation so we must define the second and third months of that quarter. We can do this by incrementing our date macro variable. Remember, in order to use the date functions we have to make sure our date macro variable is a SAS date, so we must first informat it properly:

```
%let formatdate=%sysfunc(inputn(&date., YYMMN6.));
```

Now we can increment our dates:

```
%let Month2=%sysfunc(intnx(month, &formatdate, 1));
%let Month3=%sysfunc(intnx(month, &formatdate, 2));
```

To view our new SAS dates in a format we can understand, we must format them by using the PUTN function:

```
%let formatMonth2=%sysfunc(putn(&Month2., YYMMN6.));
%let formatMonth3=%sysfunc(putn(&Month3., YYMMN6.));
```

We want to view our dates to verify that they are correct, so we use a %put statement:

```
%put My dates are &date.,&formatMonth2., &formatMonth3. and their SAS
dates are &formatdate., &Month2., &Month3.;
```

Which returns in the log:

```
My dates are 201504, 201505, 201506 and their SAS dates are 20179, 20209,
20240
```

Let's throw another little nugget in there and only do calculations if your date increments are correct – there are 2 months separating the first and last months of the quarter. We can use the INTCK() date function to define a macro variable called 'nummonths':

```
   %let nummonths=%sysfunc(intck(month, &formatdate., &month3.));
```

For some extra fun, we can then use the 'nummonths' macro variable with the conditional macro logic to check that our date increments are correct and only run our data processing if we indeed have 2 months in between the first and last month of the quarter:

```
   %if &nummonths. = 2 %then%do;
...
   %end;
      %else%do;
      %put you do not have enough dates check your data or code;
      %end;
```

Let's put our fun new macro together!

```
   %macro funwithdates;
      %let formatdate = %sysfunc(inputn(&date., YYMMN6.));
      %let Month2 = %sysfunc(intnx(month, &formatdate, 2));
      %let Month3 = %sysfunc(intnx(month, &formatdate, 3));

      %let formatMonth2 = %sysfunc(putn(&Month2., YYMMN6.));
      %let formatMonth3 = %sysfunc(putn(&Month3., YYMMN6.));

      %put my dates are &date.,&formatMonth2., &formatMonth3. and their SAS
      dates are &formatdate., &Month2., &Month3.;

      %let nummonths = %sysfunc(intck(month, &formatdate., &month3.));

      %if &nummonths. = 3%then%do;
         data NetIncome_1;
            set IncomeStatement;
                  where Date>=&formatdate.AND Date<=&Month3.;
         run;

         proc transpose data = NetIncome_1 out = NetIncome_2;
         var Value;
            by Date;
            id Operation;
         run;

         proc sql;
            create table NetIncome_3 as
   select date, (sum(Selling) + sum(Admin) + sum(GdsSold)) as TotalExpense,
   (sum(Sales)-(calculated TotalExpense)) as NetIncome
                  from NetIncome_2
            quit;
      %end;
      %else%do;
      %put you do not have enough dates, check your data or code;
      %end;
   %mend;
      %funwithdates
```

Now we have a value for the total expenses and net income for the second quarter of 2015. They are summarized in Figure 3 below:

	Date	TotalExpense	NetIncome
1	201504	262500	52500
2	201505	262500	52500
3	201506	262500	52500

Figure 3. Data set values for Net Income_3

CONCLUSION

SAS dates can be difficult to understand and challenging to handle in macro processing. However, the use of SAS dates and date functions in macro programs can greatly improve the efficiency of our code. We have covered the basics of macro programming and explored the formatting, display and processing of SAS dates, so you should now be confident that when working with these two concepts, you will never have a bad date again!

REFERENCES

Carpenter, Art. 2004. *Carpenter's Complete Guide to the SAS Macro Language*, 2nd ed. Cary, NC: SAS Institute, Inc.

Morgan, Derek. 2006. *The Essential Guide to SAS Dates and Times*.Cary, NC: SAS Institute, Inc.

CONTACT INFORMATION

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

Kate Burnett-Isaacs
Statistics Canada
Kate.Burnett-Isaacs@canada.ca

Andrew Clapson
MD Financial Management
Andrew.clapson@cma.ca