

## Paper 075-2010

**Automating Report Dates and Formats Using SAS®9 Software**

John Simeoni and Dikki Coy, Defense Logistics Agency Office of Operations Research and Resource Analysis (DORRA), Richmond, VA

**ABSTRACT**

In many organizations, analysts manually change the SAS® code that is used to run routine reports so that it uses current dates. However, analysts can use SAS date functions and formats to create automated macro variables that update all of the dates in a report script. Using macro date variables eliminates the need to manually edit scripts, and these variables can even be used to find current external files without searching for them. Date macro variables enable an analyst to execute report scripts at pre-designated times without ever having to edit the file. This ability can greatly reduce processing time and can also eliminate user error.

**INTRODUCTION**

Do you run routine monthly reports? Many analysts run regular periodic reports. Typically, these are produced monthly, but they can be scheduled for any time interval. In many organizations, analysts manually change the dates in the SAS code before running the reports. For example, a programmer will open the code, change Aug09 to Sep09, save the code, and run the program. The code often contains many instances where dates need updating, and may even have date references in several different formats. For example, here is a macro call from a piece of code that I recently inherited:

```
%getdata('2009aug01'd', 20090801, '01-AUG-2009', '31AUG2009', 200908)
```

Manual date changes are often unnecessary. We can usually automate all date references in any SAS report script. This can be accomplished by using SAS date functions, followed by proper formatting, to read the date into a macro variable. The macro variable will only depend on the run date to set all of the dates in the program. The use of automatic date macro variables alleviates the need to manually change dates (which, of course, may suffer from fat fingers), and allows the analyst to schedule the exact same script to execute month after month. With "auto-run" jobs, it is even possible to completely eliminate the need to touch the program. Automating SAS dates decreases user interface time and reduces the probability of user error.

This paper will address the following:

1. Establishing a standard nomenclature for date macro variables
2. Creating and formatting data macro variables
3. Common uses for date macro variables
4. Using date macro variables in non-owned data sets

## ESTABLISHING A STANDARD NOMENCLATURE FOR DATE MACRO VARIABLES

In most organizations, code is shared and passed on to new owners. How many times have you inherited code with macro variables named "&date1", "&date2", etc., but have no idea what they are until you see how they're resolved in the log? The definition may be buried deep in the script, or may even be pulled from another program via a "%INCLUDE" statement. Wouldn't it be great if you could determine everything about a date just from its name? Then why not have a standard naming convention for date macro variables?

A standard naming convention can be anything that is logical and documented. Once an organization creates a standard, any analyst can look at any date macro variable in any program and easily determine what it means. For example, in monthly reports, code often refers to the beginning or end of a month prior to the current one. A standard naming convention needs to clearly define the macro variable date. The macro date will be variable and based only on the run date of the program. Here is an example:

Macro Variable Date Name = **M X X D X**

### Where

- Digit #1 = M (to represent "month")
- Digit #2 = X (integer designating past, current, future)
- Digit #3 = X (integer designating the specific month)
- Digit #4 = D (to represent "day")
- Digit #5 = X (integer or letter designating the specific day)

**Digit #2** can contain the following codes:

- P = Prior (to designate a past month)
- C = Current Month
- F = Future Month (to designate a future month)

**Digit #3** can contain the following:

- X = Any integer to denote the number of months prior to the current month if Digit #2 = P
- X = 0 if Digit #2 = "C"
- X = Any integer to denote the number of months in the future if Digit #2 = F

**Digit #5** can contain the following codes:

- B = Beginning (the first day of the month)
- M = Middle (mid-month)
- E = End (the last day of the month)
- S = Same Day
- X = Any integer to denote a specific month date

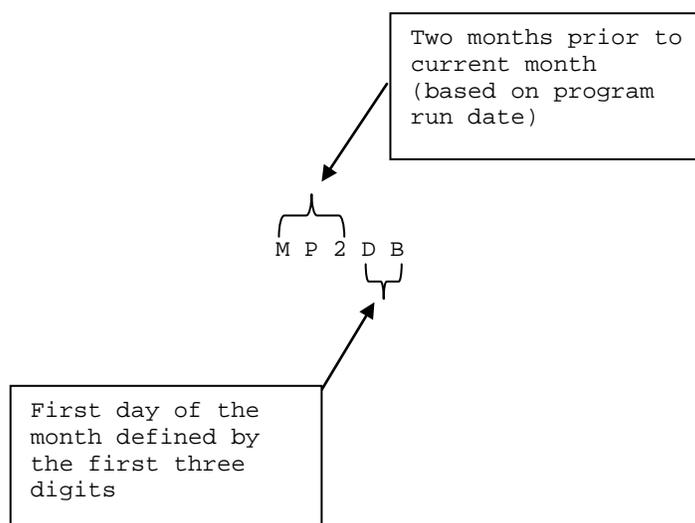
NOTE: In this format, Digit #1 is always "M" and Digit #4 is always "D". Although redundant, they help convey the meaning of the date translation.

**Example 1:**

Assume today's date is 10 September 2009 and you want to reference 01 July 2009 in a SAS program.

The example date variable format to identify 01 July 2009 is: **MP2DB**

This defines a date equal to the first day of the month two months prior to the current month.



**Figure 1.** Macro Variable Definition

**Example 2:**

Again, assume today is 10 September 2009. Here are some dates and their definitions using the sample standard nomenclature:

- |    |                   |       |   |
|----|-------------------|-------|---|
| a) | 31 July 2009:     | MP2DE | (2 months prior, last day of the month) |
| b) | 1 September 2009: | MC0DB | (Current month, first day of the month) |
| c) | 10 August 2009:   | MP1DS | (1 month prior, same day of the month)  |

Although a standard naming convention is not necessary, it will promote better understanding of the macro dates by all users.

## CREATING AND FORMATTING DATE MACRO VARIABLES

### CREATING THE DATES

SAS has built in date functions to perform date arithmetic. One of the most useful is the INTNX function, which has the following syntax:

**INTNX**(*interval, start-from, increment, alignment*)

Interval defines the time interval for date arithmetic (month, week, etc.)

Start From (for automating reports) will typically be the date the program is run, because the other dates will depend on that date.

Increment is the number of intervals being evaluated

Alignment is the specific part of the interval needed

B = Beginning  
E = End  
M = Middle  
S = Same Day

Example:

Consider the date variables created in Examples 1 & 2 above. These would be created as follows:

```
DATA TEMP ;  
  MP2DB = INTNX('month',today(),-2,'B');  
  MP2DE = INTNX('month',today(),-2,'E');  
  MC0DB = INTNX('month',today(),0,'B');  
  MP1DS = INTNX('month',today(),-1,'S');  
RUN;
```

The values created by these variables on 10 September 2009 will be the same on 11 September 2009 except for the last one, which will change to 11 August 2009. The first three won't change values until 1 October 2009. All of these variable values will remain unchanged until the first day of the next month.

NOTE: I have used the "today" function, but this can just as easily be created using the "SYSDATE" macro variable.

## FORMATTING THE DATES

The automated dates are defined above. However, you may need to format them. The dates as defined are simply SAS dates (i.e., stored as a number) that depend on the date a script is run. Any date can be formatted each time a format is required, or you can create separate macro variables for each format. We'll take the latter approach. Let's consider our variable MP2DB created above. Now, assume the following are needed:

```
01Jul2009
20090701
200907
```

We will base the macro on the standard date definition defined earlier. The macro variable will be created adding an extension to the date definition to denote the format. The actual extension can reflect the formatting explicitly (e.g., format=YYYYMMDD), although that will create long variables. An alternate approach would be to develop a template of standard formats commonly used, and add extensions to reflect them (e.g., F1, F2 to denote "format 1", "format 2"). Here, we will take the former approach. For the date formats above, we will need the following SAS formats:

```
date9.
yymmddn8.
yymmn6.
```

Thus, we want to create the following macro variables for our program:

```
MP2DB_date9          = 01JUL2009
MP2DB_yymmddn8      = 20090701
MP2DB_yymmn6        = 200907
```

## CREATING THE DATE MACRO VARIABLES

To create the macro variables, we will need to use the date definition, a put function to convert it to a character value, and a "call symput" to create the macro variable. We will create these in a \_NULL\_ data step. Thus, we have the following:

```
DATA _NULL_;
  MP2DB = INTNX('month',today(),-2,'B');
  CALL SYMPUT('MP2DB', MP2DB); /* IF NEEDED. THIS WILL BE A NUMBER */
  CALL SYMPUT('MP2DB_date9', CATS(PUT(MP2DB,date9.)));
  CALL SYMPUT('MP2DB_yymmddn8', CATS(PUT(MP2DB,yymmddn8.)));
  CALL SYMPUT('MP2DB_yymmn6', CATS(PUT(MP2DB,yymmn6.)));
RUN;
```

Verify the macro variables by submitting the %PUT \_USER\_ Statement:

```
MP2DB 18079
MP2DB_date9 01JUL2009
MP2DB_yymmddn8 20090701
MP2DB_yymmn6 200907
```

Sometimes, the macro variable must be in quotes. If so, we can add a "Q" at the end of the formatting extension, and add the quote to the CATS function. (Note: If the quotes can be double quotes, then this can be accomplished with the BQUOTE function.) Thus, '20090701' would be represented by CATS("'", PUT(MP2DB, yymmddn8.), "'");

The following would be added to the data step:

```
DATA _NULL_;
  MP2DB = INTNX('month',today(),-2,'B');
  CALL SYMPUT('MP2DB_yymmddn8q', CATS("'", PUT(MP2DB, yymmddn8.), "'"));
RUN;

%PUT _USER_;
```

Output of %PUT \_USER\_ Statement:

```
MP2DB_yymmddn8q `20090701`
```

The "CATS" function concatenates the text strings and compresses the variable by removing blanks. If using a version prior to SAS 9.2, the previous statement can be created using CONCATENATE and TRIM functions.

## COMMON USES FOR DATE MACRO VARIABLES

### USING THE DATE MACRO VARIABLES

After creating the macro variables, they can be used in many ways in the program. A benefit of using a standard methodology is that anyone that reads your code will know exactly what date you are referring to and what format you want.

Macro inputs: Dates in specific formats are often called in macros. If you inherit code, you may need to pass a date in a specific format or rewrite the code.

Data set file extensions: It is common to produce data sets used by others in the organization. If these data sets have an extension (in a particular data format), the file extension will need to be produced the same way because others' code depends on it.

#### Example:

```
DATA teamdir.myreport_&MP2DB_yymmddn8;
    SET monthlycrunch;
RUN;
```

This produces a file in the teamdir directory named myreport\_20090701

Output file extensions: As above, files may have been produced historically in a specific standard format with a date extension.

#### Example:

```
PROC EXPORT DATA = monthlycrunch
    outfile = "myreport_&MP2DB_yymmddn8..csv"
    dbms=csv
    replace;
RUN;
```

This produces a file named myreport\_20090701.csv

## EXTENDING TO RUN PRIOR MONTHS' REPORTS

Could you use the date macro variables if you had to rerun a prior month? For example, suppose in September 2009 you have a need to rerun the report from February 2009. A simple modification will enable you to make one variable change, and all other variables will update as needed. Assume you have the following macro variable data set in your "normal" report, which runs data for the previous month (i.e., runs August's report in September).

```
DATA _NULL_;
    MP1DB = INTNX('month',today(),-1,'B');
    CALL SYMPUT('MP1DB_date9',CATS(PUT(MP1DB,date9.)));
    CALL SYMPUT('MP21B_yymmn6',CATS(PUT(MP1DB,yymmn6.)));
RUN;
```

Now, define a new macro variable named "rptdiff" that replaces the "-1" in the MP1DB definition. The following is identical to the original definition:

```
%LET rptdiff = 1;

DATA _NULL_;
  MP1DB = INTNX('month',today(),0-&rptdiff,'B');
  CALL SYMPUT('MP1DB_date9', CATS(PUT(MP1DB,date9.)));
  CALL SYMPUT('MP21B_yymmnn6', CATS(PUT(MP1DB,yymmnn6.)));
RUN;
```

If you want to run February's report, you'll need to go back six months, so simply **add 6** to &rptdiff and the macro variables will update accordingly. Be sure to define all date macro variables in the program to &rptdiff. The following data set would establish dates for a February 2009 report:

```
%LET rptdiff = 7;

DATA _NULL_;
  MP1DB = INTNX('month',today(),0-&rptdiff,'B');
  CALL SYMPUT('MP1DB_date9', CATS(PUT(MP1DB,date9.)));
  CALL SYMPUT('MP21B_yymmnn6', CATS(PUT(MP1DB,yymmnn6.)));
RUN;
```

NOTE: If you need to rerun several months of reports, you can put your entire program in a macro, then run it by changing the value of &rptdiff and using a DO loop.

```
%LET begmonth = 1;
%LET endmonth = 6;
%LET rptdiff = &begmonth;

%MACRO runmany;
  %DO I = &begmonth %TO &endmonth;

    DATA TEMP;
      MPXDB = INTNX('month',today(),0-&rptdiff,'B');
      CALL SYMPUT('MPXDB', PUT(MPXDB, yymmnn6.));
    RUN;

    %LET rptdiff=%EVAL(&rptdiff+1);

    /**** BEGIN ALL OF YOU PROGRAM HERE ****/
    DATA temp_&MPXDB; /* THIS HERE JUST TO SHOW MACRO VARIABLE USE */
      junk+1;
    RUN;
    /**** END YOUR PROGRAM ****/

  %END;
%MEND;

%runmany
```

## USING DATE MACRO VARIABLES IN NON-OWNED DATA SETS

### FINDING CURRENT NON-OWNED DATA SETS

It is difficult to fully automate a script if you don't control all of the input data sets. For example, what if your SAS script uses a file created by another user, and you typically don't know the most current version of the file? For example, assume someone creates a file called prices\_MMMYY.sas7bdat, but the file is updated irregularly. Even if you run the automated program, you must first check the appropriate directory to determine the latest version of the pricing data file, and then go in and manually change the file reference in your program. However, you can automate this procedure also by using the SAS macro date variable methodology combined with a SYSFUNC(exist) function.

Let's say that the data set you need is in a folder named "pricing" and you're running a report in September, 2009. We will first check to see if there is a data set named prices\_sep09. If we can't find that, we'll look for prices\_aug09, then prices\_jul09, etc., until we find the latest one. Here is how we can do that:

Recall the macro variable technique used. We will produce a series of macro variables going as far in the past as we believe we need to (in this example, 6 months). These macro variables will be named AUG09, JUL09, etc. The program sets up an indicator variable named "tester", that increments by 1 as soon as it finds a file, and thus stops the data loop.

```

/* THIS WILL CREATE A SERIES OF VARIABLES */
DATA junk;
  DO count = 0 TO 6;
    p_date=PUT(intnx('month',today(),0-count,'B'), monyy5.);
    OUTPUT;
  END;
RUN;

/* READ INTO MACRO VARIABLES */
DATA _NULL_;
  SET junk;
  cnt=CATS(PUT(count,$2.));
  CALL SYMPUT('getdate' || cnt, p_date);
RUN;

/* FIND THE MOST RECENT ONE */
%MACRO getpricefile;
  %GLOBAL myfile;
  DATA _NULL_;
    tester=0; /*an indicator variable*/
    %DO i = 0 %TO 6;
      IF tester=0 THEN DO;
        IF %SYSFUNC(exist(pricing.prices_&&getdate&i)) THEN DO;
          CALL SYMPUT('myfile', "&&getdate&i");
          tester+1;
        END;
      END;
    %END;
  RUN;
%MEND;

%getpricefile

/* DELETE MACRO VARIABLES & CHECK THE LOG TO SEE THE CURRENT PRICE FILE */
%SYMDEL getdate6 getdate5
        getdate4 getdate3
        getdate2 getdate1
        getdate0;

%PUT _user_;

```

Once the need for the macro variables has passed, it's best to eliminate them and clean up the global macro table. The macro variable we need, &myfile, will be the only one that remains.

Contents of "Junk":

COUNT	p_date
0	SEP09
1	AUG09
2	JUL09
3	JUN09
4	MAY09
5	APR09
6	MAR09

Contents of log from %PUT \_USER after \_NULL\_ data step:

MACRO VARIABLE	VALUE
getdate0	SEP09
getdate1	AUG09
getdate2	JUL09
getdate3	JUN09
getdate4	MAY09
getdate5	APR09
getdate6	MAR09

Contents of log from %PUT \_USER after macro getpricefile:

MACRO VARIABLE	VALUE
myfile	JUL09 /* (the most recent file) */

## CONCLUSION

Automated date macro variables can greatly simplify running routine reports. By creating a standard naming convention, the specific date references and date formats will be easily understood by everyone that uses the scripts. Automating the scripts will reduce errors and require less maintenance time. Automated dates can be used to find files created by other users, and simplify running reports of multiple months of data. Fully automated scripts can be kicked off remotely at pre-designated times each month, eliminating the need to even touch the program.

## REFERENCES

- SAS Institute Inc. 2009. *SAS® 9.2 Language Reference: Dictionary, Second Edition*. Cary, NC: SAS Institute Inc.
- SAS Institute inc. 2009. *Base SAS® 9.2 Procedures Guide*. Cary, NC: SAS Institute Inc.
- SAS Institute inc. 2009. *SAS® Macro Programming: Advanced Topics*. Cary, NC: SAS Institute Inc.

## ACKNOWLEDGMENTS

Our thanks go out to Rob McAfee and Mark Jordan from SAS, who helped us on how to use macros to find files and directories.

## CONTACT INFORMATION

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

Name: John Simeoni  
Enterprise: Defense Logistics Agency; J3/4-DORRA  
Address: 8000 Jefferson Davis Hwy  
City, State ZIP: Richmond, VA 23474  
Work Phone: (804) 279-2474  
Fax: (804) 279-2292  
E-mail: [john.simeoni@dla.mil](mailto:john.simeoni@dla.mil)

Name: Dikki Coy  
Enterprise: Defense Logistics Agency; J3/4-DORRA  
Address: 8000 Jefferson Davis Hwy  
City, State ZIP: Richmond, VA 23474  
Work Phone: (804) 279-5491  
Fax: (804) 279-2292  
E-mail: [dikki.coy@dla.mil](mailto:dikki.coy@dla.mil)

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.