SESUG Paper 075-2021

Tips for Input/Handling of Dates in Command Files and Production

Thomas E. Billings, MUFG Union Bank, N.A., San Francisco, California

ABSTRACT

Production programs often have complex data handling, and this makes it difficult to run a process (often outside production) for dates other than the current period. Here we illustrate select methods to make date handling more flexible. Many production programs calculate the target interval starting with the TODAY function for the current date. We show a very simple code change that supports the option to run for alternate start dates. Checking for the last workday of the month can be messy as holidays may occur; we show a simple method that avoids this issue. Dates may be input using PROC IMPORT from an Excel® command file and the dates may come in as character variables instead of dates. We discuss ways to avoid/handle this. Date parameters may be passed via the operating system SAS® command invocation. We show easy ways to parse the values and test/set the values using SYSPARM-related features.

Keywords: SYSPARM, dates, date functions, PROC IMPORT, INTNX, TODAY
SAS® products: Base SAS.
User level: intermediate.

MAKING STANDARD CODE MORE FLEXIBLE

The date processing for production programs can be complex or simple, depending on the code and circumstances. Such programs may need to run for the preceding month, week, calendar day, quarter, or workday. We shall address workday in a later section; for the others it is common to use the TODAY and INTNX functions to define the previous calendar day, and/or previous begin and end dates of the preceding month/week/etc. INTNX can work with dates or datetimes, and this remark also applies to datetimes.

Such code – i.e.., using TODAY as start date – is not very flexible as it will run only the latest period. This can be an obstacle if one needs to run the program for a different time period, e.g., for n>1 months ago. One common solution is to make the target time period a macro variable, but then the variable needs to be set for every run, and that is not practical for fully automated production.

However, there is a way to have TODAY as the standard start date, while having the flexibility of using a different start date only when needed. To do this, define a macro variable, &OVERRIDE_DATE, which will normally be missing, but when set to a value will be used instead of TODAY. The logic is shown below for date code that derives the begin and end dates of the preceding month.

```
%global override_date;
*******************************************************************************;
%*  OVERRIDE DATE  - OPTION;
%* set this variable to a sas date (date9. format, no quotes or d) to override
default;
%* to start processing with a target month, set override_date to a day in the 1st
month;
%* AFTER the target month;
%let override_date=;           *  comment out this line or next, set date if needed;
%let override_date=14JUN2020;
*******************************************************************************;

*  set date variables;

data _null_;
      format period_end date9.;
      length yyyymm $6.;

      if ( not (missing(strip(symget("override_date")))) ) then
            do;
                  period_end=intnx("month","&override_date."d,-1,"E");
                  period_begin=intnx("month","&override_date."d,-1,"B");
            end;
      else
            do;
                  period_end=intnx("month",today(),-1,"E");
                  period_begin=intnx("month",today(),-1,"B");
            end;

      yyyymm = cats(put(year(period_end),z4.),put(month(period_end),z2.));
      call symputx("period_end",put(period_end,date9.),G);
      call symputx("period_begin",put(period_begin, date9.),G);
      call symputx("yyyymm",yyyymm,G);;
run;

%put period_begin=&period_begin;
%put period_end=&period_end;
%put yyyymm=&yyyymm.;
```

The above code runs and sets the target month dates to those for May 2020:

```
62         %put period_begin=&period_begin;
period_begin=01MAY2020
63         %put period_end=&period_end;
period_end=31MAY2020
64         %put yyyymm=&yyyymm.;
yyyymm=202005
```

In standard production, macro variable &OVERRIDE_DATE will be missing (the 1st %LET above], so the SYMGET will return a missing value, causing production to use the TODAY function for calculating start date. If &OVERRIDE_DATE has a value, then it will be used for calculating the target month begin/end dates. Savvy readers will note that the code above could be more concise if desired. (For clarity/exposition purposes we present it in the form above.)

DETERMINING THE LAST WORKDAY IN A PERIOD

Getting the last day of a calendar month (or week/etc.) is easy; just use the INTNX function. One might think that one could identify the last day of the period that is not a weekend day and use that as the last workday. However, holidays can make that problematic. SAS® has multiple HOLIDAY functions, but the code to use these can be less than elegant. Finally, checking for weekends and holidays won't identify unscheduled downtime caused by problems, system changes, and other externalities.

A relatively reliable way to identify the last work day – that does not depend on checking for weekends or holidays – is available if your data warehouse has tables that are updated only for as-of dates (or datetimes) that are work days. This method also adjusts for cases where the data warehouse is down for upgrades or other reasons.

Below is sample code that checks for the last datetime in a specific month, for a target table. It uses a macro variable (**&curr_mo_begin.**) that is previously set as the 1st day of the month following the target month, with hours, minutes, seconds all set to zero.

```
*  need datetime of last day with data in preceding month.;
*  load into macro variable;

PROC SQL;
   CREATE TABLE WORK.max_date_target_table AS
   SELECT /* max_period_dt */
            (MAX(t1.PERIOD_DT)) FORMAT=DATETIME20. LABEL="max_period_dt" AS
max_period_dt
      FROM mylib.target_table t1
      WHERE t1.PERIOD_DT < "&curr_mo_begin."dt;
QUIT;

data _null_;
       set WORK.max_date_target_table;

       if missing(max_period_dt) then
             abort cancel;
       call symputx("last_month_date",put(max_period_dt,datetime20.),"G");
run;
```

Using SQL SELECT INTO with a FORMAT could potentially make the DATA _NULL_ step above redundant. That is true if SQL will write the macro variable into the appropriate symbol table (local or global). The DATA _NULL_ approach allows more control over which symbol table is used.

This method has potential disadvantages and should be thoroughly tested before adopting it for a particular program. You want to use a database table that is updated only on workdays. Some database tables are massive, and the query above might take a very long time to run. If that happens, possible mitigations are:

- Switch to a smaller table – perhaps a staging table, or
- Choose a table where PERIOD_DT is indexed (or a key/part of a key), or
- Modify the test on PERIOD_DT so that it is checking the parameter only over a specified short period, e.g. for a last workday in a month., check the period: [month_end - 5 days, month_end].

EXCEL® COMMAND FILES: VARIABLES ARE THE WRONG TYPE

Command files are sometimes used to control (production) programs. These are usually small files that contain parameters that identify the work to be done by the target program. If these are Excel files are input via PROC IMPORT, then if an input parameter has no value in the command file, it may be imported as a character variable instead of a numeric or date variable. This can be a problem when a date/datetime field is optional and left unfilled in a command file that has a few (or 1) lines of real data. Obviously, this can break downstream logic and cause your program to fail.

There are multiple ways to deal with this. Some of the ways are:

1. Switch from Excel to a flat file input: delimited, formatted, XML, etc., input via a DATA step;
2. Use PROC IMPORT and in follow-up DATA steps, check the variable types, convert if necessary
3. Use the PROC IMPORT options (not available for all DBMS= options) DBDSOPTS with DBSASTYPE= option.
4. Use a LIBNAME with the EXCEL data engine and the DBSASTYPE= option.

Method #2 works but is labor intensive if there are several variables with the issue. To use it, have a follow-up DATA step with a SET on the imported SAS file and the target variables are renamed. Then use the VTYPE or VTYPEX function with IF-THEN logic to determine if the variable was input correctly, and process accordingly. For example:

```
data work.fix_input;
       set work.import_file (rename= (run_date=run_date_in));
       length run_date 8;
       format run_date date9.;
       call missing(run_date);

       if vtype(run_date_in) = 'C' then
              do;
                     if (not missing(run_date_in)) then
                            run_date = input(strip(run_date_n),date9.);
              end;
run;
```

Method #3 is for advanced users familiar with the options for their choice of DBMS= in PROC IMPORT. For example:

DBDSOPTS= "DBSASTYPE=(run_date='DATE')";

Will force run_date to be a date (you will want to apply the relevant format afterwards).

Method #4 is similar to #3, but needs only the DBSASTYPE= option. Note that DBSASTYPE is a data set option that can be used in many other contexts; read the SAS documentation before trying to use this method. Method #1 using comma, pipe, or tab-delimited files is probably the easiest to maintain of the 3 methods. Methods 3,4 are simple but programmers are often not familiar with the options.

PASSING PARAMETERS IN SYSPARM

For batch programs that are run via the SAS operating system command, parameters can be passed in the call to the SAS program via the sysparm= option. The values passed via this method are available in programs via the SYSPARM function, the &SYSPARM macro variable (i.e., SYMGET('SYSPARM')) Further, this can be tested (outside of batch, if desired), using OPTIONS SYPARM= to set the sysparm values.

SYSPARM is a convenient method to pass parameters, including date or datetime values to a program. Many parameters are of the form:

parameter_name = parameter_value

and these need to be parsed. SAS has what is known as named INPUT, i.e., the ability to read – via the INPUT statement in a DATA step, but not the INPUT function – parameters in the form shown above. We can use this to input the target parameters; here is sample code.

```
*  Attention: this DATA step uses DATALINES4, Do NOT encapsulate;
*  this DATA step inside a macro as if you do, it might not work;

data date_parms;
     informat yyyymm $8.  start end ANYDTDTE10.;
     infile datalines4;
     input @;
     _infile_ = translate(symget('sysparm'), ' ', ',');
     input yyyymm=  start=  end=;

  *  code to do error checks on input parameters goes here;
  *  along with code to handle defaults if certain parameters are missing;

     output;
     stop;
     datalines4;

run;
;;;;
```

The INPUT @ holds the row, which is blank/missing; we then set _INFILE_ with the values of the SYSPARM parameters, after which named INPUT and assigned INFORMATS do the parsing. Note the comment that the code above might not work inside a macro.

Remark: SAS named INPUT is similar to FORTRAN namelist input and is very useful in certain contents.

EPILOLGUE

We have presented multiple methods for handling dates in programs. Hopefully some of these methods will be useful to you.

Note: As of the date of publication, the author is an employee of MUFG Union Bank, N.A. The content of this paper does not reflect the views or opinions or work product of MUFG Union Bank

APPENDIX 1:
BSD 2-CLAUSE COPYRIGHT LICENSE (OPEN SOURCE)

```
* All program code in this paper is released under a Berkeley Systems
  Distribution BSD-2-Clause license, an open-source license that permits
  free reuse and republication under conditions;

/*
Copyright (c) 2020, MUFG Union Bank, N.A.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are
permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of
conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list
of conditions and the following disclaimer in the documentation and/or other materials
provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
*/
```

CONTACT INFORMATION

A list of the author's SAS-related papers, including URLs for free access, is available at the URL (hosted by Google Drive): https://goo.gl/uCUHoa

Note: Your enterprise web filter might prevent access to this URL from work, in which case you will need to access via a personal device.

Thomas E. Billings
Email: tebillings@gmail.com