Paper 078-2009

Time Travel: How To Easily Create, Loop Through And Manipulate Data With Names Containing YYYYMM

Jingjing Shan, JPMorgan Chase, NJ

Abstract

In financial industries, the information is vast. To limit the size of each dataset, they are usually organized as monthly dataset. In many cases, the dataset name contains the date information. However, analysis and reporting often require cross time periods. For example, the monthly prepayment calculation needs information from both current and prior month. Often you will be utilizing multiple monthly data to either build models or compare projections against actual experience. Even though YYYYMM is not continues as regular numbers, you would like to have a simple way to just specify the beginning period and ending period, your programs will just gracefully loop through.

With the help of SAS® functions INTNX, TODAY, SYMPUT, PUT, %SUBSTR, %EVAL, MDY and various date formats, a simple shell macro will do the trick. In this paper, we will show to automatically create monthly data, organize and travel through them as well as a technique to extract date information out variable name based on the value of the variable without using PROC TRANSPOSE to create another large dataset.

Introduction

Most large companies have an Oracle® or DB2® based data warehouse. But they are largely treated as a data repository by many modeler and analysts. Different groups still download them every month to clean them and add flags. More importantly, the processing speed is much different as you are working off data storing locally, on LAN or SAN drives with your own PC or Unix box instead of competing resources with thousands other people in the company hitting the data warehouse at the same time at the beginning of each month.

To illustrate the techniques, let's assume we are dealing with monthly data. They are as of last month end. The same logic can be applied to other frequencies (daily, weekly, or quarterly) and other type of data.

We will illustrate how to create a dynamic program to create monthly dataset with a name such as Servicing_200809 with no need to modify any code each month and loop through a period of multiple months to process them by simply specifying beginning period and ending period, i.e. 200501 and 200809.

In this paper, we will also illustrate how a horizontal master dataset with static information such account number, date of open and dynamic information such as monthly balance, payment and status can be created from multiple monthly datasets. Each monthly measure (i.e. balance, payment, status, etc.) is in one variable, not stacked in one variable and identified by a corresponding as_of_date field as in a relational database table -- creation of this type of horizontal master dataset is often part of initial steps of data preparation for model building.

We will also demonstrate a technique to extract date information from variable name (i.e. Balance_200801, Balance_200802, …, Balance_200809) based on the value of the variables (i.e, during which month the balance is higher than prior month or over a specific amount, that date information is in the variable name not in the data). Or to track how many and after how many months the modified delinquent mortgage loans will be delinquent again. Our technique will not need to create another large data set with PROC TRANSPOSE.

The data

Here is a simplified example of data in an Oracle database

AS_OF_DATE	LOAN_ID	BALANCE	STATUS

30SEP2008	1	100000	0
30SEP2008	2	200000	0
30SEP2008	3	300000	0
30SEP2008	4	400000	0
30SEP2008	5	500000	0
30SEP2008	6	510000	30
30SEP2008	7	520000	30
30SEP2008	8	530000	60
30SEP2008	9	540000	90
30SEP2008	10	550000	90

Task 1: Create Them

To download the month end information of prior month to a SAS dataset with name of Servicing_YYYYMM.

High maintenance way:

Program 1:

```
OPTION MPRINT;

%LET USER_ID=myID;
%LET PW=myPassword;
%LET YYYYMM=200809;
%LET MMDDYYYY='0932008';

LIBNAME myDIR 'C:\MonthlyData';

%MACRO GET_DATA(OUT_DIR, MMDDYYYY, YYYYMM);

        PROC SQL;
        CONNECT TO ORACLE (USER=&USER_ID PASSWORD="&PW" PATH='myPath');
        CREATE TABLE &OUT_DIR..Servicing_&YYYYMM AS
        SELECT * FROM CONNECTION TO ORACLE
        (
                SELECT
                Loan_ID
                ,Balance
                ,Status
                ,As_of_Date

                FROM  mySchema.LOANS

                WHERE
                        AS_OF_DATE = TO_DATE(&MMDDYYYY.,'MMDDYYYY') AND

        );
        DISCONNECT FROM ORACLE;
        QUIT;

%MEND GET_DATA;

%GET_DATA(myDIR, &MMDDYYYY, &YYYYMM);
```

Every month you need to update two macro variables – it is not too bad, as long as you remember to update them correctly – especially Februarys of leap years. If you need to go back history and download 120 months data, you probably will have a program of 120 lines like this:

```
%GET_DATA(myDIR, '10311999', 199910);
%GET_DATA(myDIR, '11301999', 199911);
.
.
```

078-2009

```
.
%GET_DATA(myDIR, ‘08312008’, 200808);
%GET_DATA(myDIR, ‘09302008’, 200809);
```

Hope you type accurately and fast. I can’t. There is a trick to use Excel to quickly generate code like this in the appendix. But a more elegant solution is to use INTNX function to generate dates you need and use SYMPUT function to assign them to macro variables.

Easier way:

Program 2

```
OPTION MPRINT;

%LET USER_ID=myID;
%LET PW=myPassword;

LIBNAME myDIR 'C:\MonthlyData';

%MACRO GET_DATA(OUT_DIR, MMDDYYYY, YYYYMM);

        PROC SQL;
        CONNECT TO ORACLE (USER=&USER_ID PASSWORD="&PW" PATH='EDW_PROD');
        CREATE TABLE &OUT_DIR..Servicing_&YYYYMM AS
        SELECT * FROM CONNECTION TO ORACLE
        (
                SELECT
                Loan_ID
                ,Balance
                ,Status
                ,As_of_Date

                FROM  mySchema.LOANS

                WHERE
                        AS_OF_DATE = TO_DATE(&MMDDYYYY.,'MMDDYYYY') AND

        );
        DISCONNECT FROM ORACLE;
        QUIT;

%MEND GET_DATA;

Data _null_;
 CALL SYMPUT('MMDDYYYY', "'"||PUT(INTNX('MONTH',today(), -1,'E'),MMDDYYN8.)||"'");
 CALL SYMPUT('YYYYMM', PUT(INTNX('MONTH',today(),- 1,'B'),YYMMN6.));
Run;

%GET_DATA(myDIR, MMDDYYYY, YYYMM);
```

Running the above program at any time of a month will always create a data set with previous month end information and the data set is with a name like Servicing_YYYYMM – you don’t need change any thing each month (you can forget about the leap year)!

The SAS function INTNX increments a date, time, or datetime value by intervals such as DAY, WEEK, QTR, and MINUTE. The increment is based on a starting date, time, or datetime value, and on the number of time intervals that you specify.

The syntax of INTNX

INTNX(INTERVAL, START-FROM, INCREMENT, <,ALIGNMENT>)

INTERVAL argument can be YEAR, SEMIYEAR, QTR, MONTH, WEEK, DAY. *START-FROM* is any date, time or datetime value. *INCREMENT* argument can be a positive, negative or zero integer that specifies the number of intervals

to shift from *START-FROM* point. Optional *ALIGNMENT* argument can be B, the return date will be aligned to the beginning of interval; E, the return date is aligned to the end of the interval; M, the return date is aligned to the middle of the interval. The default alignment is 'B'.

Since our program is run monthly for prior month end information, our interval is 'MONTH'. The start-from is *TODAY*() that returns today's system date. The increment needs to be negative to shift back one month. The alignment is 'E' to get the last day of the prior month.

SYMPUT function assigns a constant to a global macro variable:

CALL SYMPUT ('macro_variable', constant)

To correctly pass date information to Oracle in MMDDYYYY form, the date constant has to be in quotation marks. Otherwise the days between 1 and 9 will not be processed correctly by TO_DATE function due to leading zero. This can be easily accomplished by using || concatenation operator to add "'" in the front and at the end.

If you want to organize your downloaded files by year, you may use X command in Windows environment to create a YYYY directory automatically. Here is the full program to automatically download and save data in a directory of your desire – if you use it with a scheduler, you may not even have to show up for work.

Program 2A

```
OPTION MPRINT XNOWAIT;

%LET USER_ID=myID;
%LET PW=myPassword;


%MACRO GET_DATA(OUT_DIR, MMDDYYYY, YYYYMM);

        PROC SQL;
        CONNECT TO ORACLE (USER=&USER_ID PASSWORD="&PW" PATH='EDW_PROD');
        CREATE TABLE &OUT_DIR..Servicing_&YYYYMM AS
        SELECT * FROM CONNECTION TO ORACLE
        (
                SELECT
                Loan_ID
                ,Balance
                ,Status
                ,As_of_Date

                FROM  mySchema.LOANS

                WHERE
                        AS_OF_DATE = TO_DATE(&MMDDYYYY.,'MMDDYYYY') AND

        );
        DISCONNECT FROM ORACLE;
        QUIT;

%MEND GET_DATA;

Data _null_;
 CALL SYMPUT('MMDDYYYY', "'"||PUT(INTNX('MONTH',today(), -1,'E'),MMDDYYN8.)||"'");
 CALL SYMPUT('YYYYMM', PUT(INTNX('MONTH',today(),- 1,'B'),YYMMN6.));
Run;

X "MD C:\MonthlyData\&YYYY";

LIBNAME myDIR "C:\MonthlyData\&YYYY";


%GET_DATA(myDIR, MMDDYYYY, YYYMM);
```

078-2009

The option XNOWAIT tells Windows to close the Command window after creating the YYYY subdirectory. If the director already exists, Windows will just flash a message of "Directory already exists." There will be no harm done.

To get 120 months of data and store them in separate 120 monthly dataset:

Program 3

```
%MACRO RUNYYMM(BEG_YYMM,END_YYMM);
%LET BEG_YY = %EVAL(%SUBSTR(&BEG_YYMM,1,4) * 1);
%LET BEG_MM = %EVAL(%SUBSTR(&BEG_YYMM,5,2) * 1);
%LET END_YY = %EVAL(%SUBSTR(&END_YYMM,1,4) * 1);
%LET END_MM = %EVAL(%SUBSTR(&END_YYMM,5,2) * 1);

*GET ROUNDS OF THE LOOP;

%LET L=%EVAL((&END_YY-&BEG_YY)*12 + (&END_MM - &BEG_MM) + 1);

*GENERATE THE DATE CONSTANTS, THESE ARE GLOBAL VARIABLES;

%DO I = 1 %TO &L;
   DATA _NULL_;

   /* Ways to generate date MACRO VARIABLES*/

      CALL SYMPUT('YYYYMM', PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I - 1,'B'),YYMMN6.));

      CALL SYMPUT('MMDDYYYY', "'"||PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I -
      1,'E'),MMDDYYN8.)||"'");
   RUN;

*PUT THE DATE CONSTANTS IN THE LOG;
  %PUT "YYYYMM IS " &YYYYMM;
  %PUT "MMDDYYYY IS " &MMDDYYYY;
  %PUT "THIS IS LOOP " &I;

  X "MD 'C:\MonthlyData\&YYYY";

  LIBNAME myDIR "C:\MonthlyData\&YYYY";

  %GET_DATA(myDIR, MMDDYYYY, YYYMM);

%END;

RUN;
%MEND;

%RUNYYMM(199810, 200809);
```

The above program will download 120 months of data and put them in 120 separate monthly datasets under ten subdirectories. One of the advantages of downloading and storing them in separate datasets is that you will have the months you have already downloaded even if the data warehouse kills all programs during routine midnight maintenance. You can substitute %GET_DATA macro with your own and modify the date format in the SYMPUT/ PUT/INTNX function inputs to generate date macro variables to meet the requirements of your macro.

Here are some useful examples of date formats:

Assume &Beg_YY = 2007, &Beg_MM = 1 and &I = 1

```
*1. '01JAN2007'D - BEGINNING OF THE MONTH
OR '31JAN2007'D - END OF THE MONTH FORMAT;
    CALL SYMPUT('BDATE', "'"||PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I - 1,'B'),DATE9.)
    ||"'D");
```

078-2009

```
     CALL SYMPUT('EDATE', "'"||PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I - 1,'E'),DATE9.)
     ||"'D");

*2. 200701 YEARMM FORMAT;
     CALL SYMPUT('BYYYYMM', PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I - 1,'B'),YYMMN6.));
     *IF ANOTHER CONSTANT IS NEEDED FOR THE FOLLOWING MONTH, CHANGE &I - 1 TO &I;
     CALL SYMPUT('SYYYYMM',PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I ,'B'),YYMMN6.));


*3 SINGLE MONTH  1 OR 12, NOTE THERE WILL BE A LEADING SPACE FOR SINGLE DIGIT MONTH;
     CALL SYMPUT('SINGLEMM', PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I - 1,'B'),MONTH2.));

*4 SINGLE MONTH 1 OR 12, NOTE THERE WILL BE NO LEADING SPACE FOR SINGLE DIGIT MONTH;
     CALL SYMPUT('SINGLEMM2', TRIM(LEFT(PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I -
     1,'B'),MONTH2.))));

*5 MONTH WITH LEADING ZERO: 01 OR 12;
     CALL SYMPUT('LEADINGMM', PUT(MONTH(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I - 1,'B')),Z2.));

*6 '02282007' MMDDYYYY FORMAT with QUOTATION MARKS;
     CALL SYMPUT('BMMDDYYYY', "'"||PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I-
     1,'B'),MMDDYYN8.)||"'");
     CALL SYMPUT('EMMDDYYYY', "'"||PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I -
     1,'E'),MMDDYYN8.)||"'");

*7 BEGINNING OR END OF DAY OF THE MONTH: 1 OR 31;
     CALL SYMPUT('BDAY', PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I - 1,'B'), DAY2.));
     CALL SYMPUT('EDAY', PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I - 1,'E'), DAY2.));
```

To test you date format you may simply use the following program:

```
%MACRO TESTDATE(BEG_YYMM);
%LET BEG_YY = %EVAL(%SUBSTR(&BEG_YYMM,1,4) * 1);
%LET BEG_MM = %EVAL(%SUBSTR(&BEG_YYMM,5,2) * 1);

DATA _NULL_;
 CALL SYMPUT("myDate1", PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),0,'B'), DAY2.));
 CALL SYMPUT('myDate2', "'"||PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),0,'B'),DATE9.) ||"'D");
 CALL SYMPUT('myDate3', PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),0,'B'),YYMMN6.));
RUN;

%PUT "My date1 is " &myDate1;
%PUT "My date2 is " &myDate2;
%PUT "My date3 is " &myDate3;

%MEND;

%TestDate(200810);
```

Your log will look like the following:

```
"My date1 is "  1
"My date2 is " '01OCT2008'D
"My date3 is " 200810
```

Task 2 Organize Them

Now you have created all these monthly data in separate datasets, what if you have to stack them to create one big time series dataset? Following is a typical program of a novice programmer would do (to keep illustration simple, here on we assume all datasets are in one directory):

Program 4

```
LIBNAME myDIR ‘C:\MonthlyDATA’;
```

```
DATA myDIR.Servicing_199910_200809;
    SET
          myDIR.Servicing_199910
          myDIR.Servicing_199911
          .
          .
          .
          myDIR.Servicing_200809;
RUN;
```

Again, you need to be able to type fast and accurately (or use the trick outlined in the appendix). And this program is inefficient. Since SET statement in a DATA step will go through each record of each referenced dataset, it is time consuming. By using PROC DATASETS (probably one of most under utilized basic PROCs) and our tricks of SYMPUT and INTNX, we can accomplish this task much more efficiently and with much less typing:

Program 5

```
LIBNAME myDIR 'C:\MonthlyDATA';
LIBNAME mastDIR 'C:\MasterDATA';

%MACRO ORGYYMM(BEG_YYMM,END_YYMM, dNAME);
%LET BEG_YY = %EVAL(%SUBSTR(&BEG_YYMM,1,4) * 1);
%LET BEG_MM = %EVAL(%SUBSTR(&BEG_YYMM,5,2) * 1);
%LET END_YY = %EVAL(%SUBSTR(&END_YYMM,1,4) * 1);
%LET END_MM = %EVAL(%SUBSTR(&END_YYMM,5,2) * 1);

*GET ROUNDS OF THE LOOP;

%LET L=%EVAL((&END_YY-&BEG_YY)*12 + (&END_MM - &BEG_MM) + 1);

DATA _NULL_;
  CALL SYMPUT('BYYYYMM',PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),0,'B'),YYMMN6.));
  CALL SYMPUT('EYYYYMM',PUT(INTNX('MONTH',MDY(&END_MM,1,&END_YY),0,'B'),YYMMN6.));
RUN;

%DO I = 1 %TO &L;

DATA NULL;

  CALL SYMPUT('YYYYMM',PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I-1,'B'),YYMMN6.));

RUN;

  %IF &I = 1 %THEN %DO;
        PROC DATASETS LIB = WORK;
          COPY IN=myDIR OUT=mastDIR;
          SELECT &dNAME._&YYYYMM;
        RUN;

        PROC DATASETS LIB = mastDIR;
          CHANGE &dNAME._&YYYYMM = &dNAME._&BYYYYMM._&EYYYYMM;
        RUN;
  %END;

  %ELSE %DO;
    PROC DATASETS;
        APPEND BASE = mastDIR.&dNAME._&BYYYYMM._&EYYYYMM
       DATA = myDIR.&dNAME._&YYYYMM;
    QUIT;
        RUN;
  %END;

*PUT THE DATE CONSTANTS IN THE LOG;
  %PUT "YYYYMM IS " &YYYYMM;
  %PUT "THIS IS LOOP " &I;

%END;
```

078-2009

```
RUN;
%MEND;

%ORGYYMM(199810, 200809, Servicing);
```

The above program is more efficient in concatenating datasets as it uses COPY and APPEND instead of SET in a data step which runs through each record. This program can be reused for any period of time and it can be applied to any type of data as long as their names are like NamePrefix_YYYYMM. NamePrefix can be flexibly specified in the last argument of ORGYYMM macro.

Now you need to build a model. The data has to be organized horizontally:

Loan_ID	Bal_199810	Bal_199811	Bal_199812 …	Bal_200809
1	100000	99750	99700	85000
2	200000	199500	199000	180000

A slight change of Program 5 in the loop section will do the trick:

Program 6

```
LIBNAME myDIR 'C:\MonthlyDATA';
LIBNAME mastDIR 'C:\MasterDATA';

%MACRO HoriYYMM(BEG_YYMM,END_YYMM, dNAME);
%LET BEG_YY = %EVAL(%SUBSTR(&BEG_YYMM,1,4) * 1);
%LET BEG_MM = %EVAL(%SUBSTR(&BEG_YYMM,5,2) * 1);
%LET END_YY = %EVAL(%SUBSTR(&END_YYMM,1,4) * 1);
%LET END_MM = %EVAL(%SUBSTR(&END_YYMM,5,2) * 1);

*GET ROUNDS OF THE LOOP;

%LET L=%EVAL((&END_YY-&BEG_YY)*12 + (&END_MM - &BEG_MM) + 1);

DATA _NULL_;
  CALL SYMPUT('BYYYYMM',PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),0,'B'),YYMMN6.));
  CALL SYMPUT('EYYYYMM',PUT(INTNX('MONTH',MDY(&END_MM,1,&END_YY),0,'B'),YYMMN6.));
RUN;

%DO I = 1 %TO &L;

DATA _NULL_;

  CALL SYMPUT('YYYYMM',PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I-1,'B'),YYMMN6.));

RUN;

  %IF &I = 1 %THEN %DO;
    PROC DATASETS LIB = WORK;
         COPY IN=myDIR OUT=mastDIR;
         SELECT &dNAME._&YYYYMM;
       RUN;

       PROC DATASETS LIB = mastDIR;
         CHANGE &dNAME._&YYYYMM = H_&BYYYYMM._&EYYYYMM;
         MODIFY H_&BYYYYMM._&EYYYYMM;
         RENAME BAL = BAL_&YYYYMM;
         QUIT;
       RUN;

    PROC SORT DATA = mastDIR.H_&BYYYYMM._&EYYYYMM;
      BY Loan_ID;

    RUN;

    %END;
```

```
  %ELSE %DO;
    PROC SORT DATA = mastDIR.H_&BYYYYMM._&EYYYYMM;
      BY Loan_ID;
    RUN;

    PROC SORT DATA = myDIR.&dNAME._&YYYYMM;
      BY Loan_ID;
    RUN;

    DATA mastDIR.H_&BYYYYMM._&EYYYYMM;
         MERGE mastDIR.H_&BYYYYMM._&EYYYYMM (IN=IN1)
           myDIR.&dNAME._&YYYYMM (IN=IN2 KEEP=LOAN_ID BAL RENAME=(BAL=BAL_&YYYYMM));
         BY LOAN_ID;
         IF IN1 OR IN2;
       RUN;
  %END;

*PUT THE DATE CONSTANTS IN THE LOG;
  %PUT “YYYYMM IS “ &YYYYMM;
  %PUT “THIS IS LOOP “ &I;

%END;

RUN;
%MEND;

%Hori YYMM(199810, 200809, Servicing);
```

Again, PROC DATASETS is used here to copy the first dataset to the master horizontal dataset and use MODIFY and RENAME statements to update the field name of BAL to BAL_YYYYMM instead of using a DATA step, which will take time to go through each record.

By now you should be able to see the core idea on how to loop through data with name (either dataset or variable) containing YYYYMM (or any other date format!). They can be easily accomplished by using a shell MACRO, %DO loop, and SYMPUT, PUT and INTNX functions in conjunction of various date formats.

Here is the shell program for time travel:

```
%MACRO TRAVYYMM(BEG_YYMM,END_YYMM, dNAME);
%LET BEG_YY = %EVAL(%SUBSTR(&BEG_YYMM,1,4) * 1);
%LET BEG_MM = %EVAL(%SUBSTR(&BEG_YYMM,5,2) * 1);
%LET END_YY = %EVAL(%SUBSTR(&END_YYMM,1,4) * 1);
%LET END_MM = %EVAL(%SUBSTR(&END_YYMM,5,2) * 1);


%LET L=%EVAL((&END_YY-&BEG_YY)*12 + (&END_MM - &BEG_MM) + 1);

DATA _NULL_;
  CALL SYMPUT('BYYYYMM',PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),0,'B'),YYMMN6.));
  CALL SYMPUT('EYYYYMM',PUT(INTNX('MONTH',MDY(&END_MM,1,&END_YY),0,'B'),YYMMN6.));
RUN;

%DO I = 1 %TO &L;

DATA _NULL_;

  CALL SYMPUT(‘YYYYMM’,PUT(INTNX(‘MONTH’,MDY(&BEG_MM,1,&BEG_YY),&I-1,’B’),YYMMN6.));

RUN;

  %YOUR_MACRO(&YYYYMM);

  %PUT “YYYYMM IS “ &YYYYMM;
  %PUT “THIS IS LOOP “ &I;

%END;
RUN;
```

```
%MEND;
```

If your macro needs more than one date related input, you may just create them in the DATA _NULL_ step based on your needs. For example, if you need current month and prior month balance to calculate balance change, you DATA _NULL_ step in the loop will look like

```
DATA _NULL_;

  CALL SYMPUT('YYYYMM',PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I-1,'B'),YYMMN6.));
  CALL SYMPUT('YYYYMM_PRIOR',PUT(INTNX('MONTH',MDY(&BEG_MM,1,&BEG_YY),&I-2,'B'),YYMMN6.));

RUN;
```

And your macro call will be:

```
%YOUR_MACRO(&YYYYMM, &YYYYMM_PRIOR);
```

Task 3 Get information out of dataset variable names

Now you have a horizontal master dataset containing monthly delinquency status.

```
DATA Status;
INFORMAT loanID 3. Status_200611 $2.  Status_200612 $2.  Status_200701 $2. Status_200702 $2.
Status_200703 $2.;
INPUT loanID    Status_200611  Status_200612  Status_200701 Status_200702  Status_200703;
DATALINES;
  1 0  0  30 60 90
  2 30 60 90 90 0
;
RUN;
```

The value in the STATUS variables indicates the number of days delinquency, 0 for on time, 30 for 30 days over due, so on and so forth. Your task is to find the first month in which the customer had a 30 days delinquency and store that information in variable: D30_Date for later analysis. The date information is in the suffix of the variable names.

Here is how you can extract that date information:

Program 7

```
PROC CONTENTS DATA=Status OUT=Status_Contents;
RUN;

*Keep only the variables start with Status;
*Please make sure the input data is indeed created this way;

DATA Status_Var;
FORMAT Name $13.; * make sure its size is large enough for the variable name in question;
 SET Status_Contents(keep=name);
 WHERE upcase(substr(left(Name),1,6))='STATUS'; *update the SUBSTR inputs and conditions
accordingly;
 RUN;

PROC SORT DATA=STATUS_VAR;
 *sorting is important to identify the first month with 30 day delinquency;
 BY Name;
RUN;

DATA Status_Var;
 FORMAT DateInfo $7.;
 SET Status_Var;
 DateInfo = TRANWRD(Name,"Status_", "");
```

078-2009

```
RUN;

PROC SQL;
 SELECT COUNT(*)
  INTO :VarCount
 FROM Status_Var;

 SELECT NAME
  INTO :NAMELIST SEPARATED BY ' '
 FROM Status_Var;

 SELECT DateInfo
  INTO :DateLIST SEPARATED BY ' '
 FROM Status_Var;
QUIT;

DATA STATUS_VAR3(DROP=I);
FORMAT D30_Date YYMMN6.;
 ARRAY STATUS(&VarCount) $ &NameList;

 *_TEMPORARY_ option allows the array to point to constant values instead of referring
variables in the dataset;
 ARRAY DATES(&VarCount)  _TEMPORARY_ (&DateList);

 SET STATUS;

DO I= 1 TO &VarCount BY 1;
   IF STATUS{I}='30' THEN DO; *Change to whatever criteria for matching, watch out format,
you will make a mistake here and complain;
      D30_Date = MDY(SUBSTR(LEFT(DATES{I}),5,2),1, SUBSTR(LEFT(DATES{I}),1,4));
      LEAVE; * Get out of Do-Loop;
   End;
End;
RUN;
```

The above program uses PROC CONTENTS to generate an output dataset containing the variable list. The variables with suffix of STATUS are put in one small dataset STATUS_VAR. This dataset has one variable NAME only: it contains the variable names which has suffix of STATUS of the original horizontal master dataset.

PROC SQL's SELECT INTO: is used to assign the values of variable NAME and DateInfo to two global macros variables: &NameList and &DateList. Each element is separated by a space. They are then used for creating arrays in the last step.

The content of NameList:

Status_200611 Status_200612 Status_200701 … Status_200703

The content of DateList is:

200611 200612 200701 … 200703

The first ARRAY of STATUS is a typical array. It refers the Status_YYYYMM variables in the original STATUS dataset. The second array's elements are not pointing to any variables. Their elements are actually the value of the DateList, i.e, the value of DATE{1} is '200601', it is a constant. Not referring to a variable with name '200601' in dataset STATUS. This is done with the _TEMPORARY_ option when the array is defined. Effectively, we extracted the date information out of the variable name.

In the last loop we can easily find the first Status_YYYYMM variable which contains a value of 30. We then use the array subscript I to identify the corresponding date information from DATE array and assign it to the D30_Date variable.

With slight modifications to the loop part, this program can be adapted to identify the number of months from origination when the first 30 delinquency happened, the likely hood of ever delinquency ending up with foreclosure, etc. For a dataset with balance information of home equity line of credit (HELOC), you can use this program to find out how

many accounts start to use their line in the first three months, at which month (on book) their balances peak, when they may pay down the total balance, once they pay down the balance, how often if ever they are using the line again and how long is the interval between them. All these identifications are important in building delinquency transition models or HELOC utilization models. They can also be used to track the effectiveness of marketing / loss prevention / loss mitigation programs easily.

By using PROC TRANSPOSE to create a stacked time series dataset, similar tasks can also be accomplished with PROC SQL and complex subquery techniques. It would indeed be complex and more time consuming to run for large datasets.

Assume there monthly balance variables in the original dataset: Balance_200611, Balance_200612, …, Balance_200703. Use the technique outlined above, we can create another global macro variable BALANCE. Its content looks like:

```
Balance_200611 Balance_200612 … Balance_200703
```

A new array needs to be defined as

```
ARRAY BALANCE(&VarCount) $ &BALANCE;
```

For a HELOC account (balance can fluctuate like a credit card), to identify the month in which month the first use occurred, the month in which the balance is the highest, in which month the balance was paid down and in which month usage happened again after paydown, the loop portion would look like the following:

```
FirstUse = .;
MaxBal = .;
MaxBalDate = .;
PayDown = .;
UseAfterPayDown = .;

DO I = 1 TO &VarCount;

  IF (Balance{I} > 0 and FirstUse =.) THEN DO;
     FirstUse = Balance{I};
     FirstUseDate =  MDY(SUBSTR(LEFT(DATES{I}),5,2),1, SUBSTR(LEFT(DATES{I}),1,4));
  END;

  IF Balance{I} >  MaxBal THEN DO;
      MaxBal = Balance{I}
      MaxBalDate = MDY(SUBSTR(LEFT(DATES{I}),5,2),1, SUBSTR(LEFT(DATES{I}),1,4));
  END;

  IF (MaxBal >0 and Balance{I} <= 0 and PayDown = .)  THEN DO;
      PayDown = 1;
      PayDownDate = MDY(SUBSTR(LEFT(DATES{I}),5,2),1, SUBSTR(LEFT(DATES{I}),1,4));
  END:

  IF (MaxBal >0 and PayDown = 1 and Balance{I} > 0  and UseAfterPayDown = .) THEN DO;
      UseAfterPayDown = 1;
      UseAfterPayDownDate = MDY(SUBSTR(LEFT(DATES{I}),5,2),1, SUBSTR(LEFT(DATES{I}),1,4));
  END:


END;

FirstUseMonOnBook = INTCK('month', OpenDate, FirstUseDate) + 1;
MaxMonOnBook = INTCK('month', OpenDate, MaxBalDate) + 1;
PayDownMonOnBook = INTCK('month', OenDate, PayDownDate) + 1;
UseAfterPayDownOnBook = INTCK('month', OenDate, UseAfterPayDownDate) + 1;
```

FirstUse will have the balance of the month in which the first time usage occurred. If it is null, there is no use during the period the data is available. MaxBal will contain the maximum balance amount and MaxBalDate will have the month in which the balance was the highest. MaxMonOnBook is the number of month interval between the date the account was opened and the highest balance month (OpenDate is the date at which the account was opened). INTCK function measures the interval between two dates. The intervals can be days, weeks, months, quarters and years. If there is no

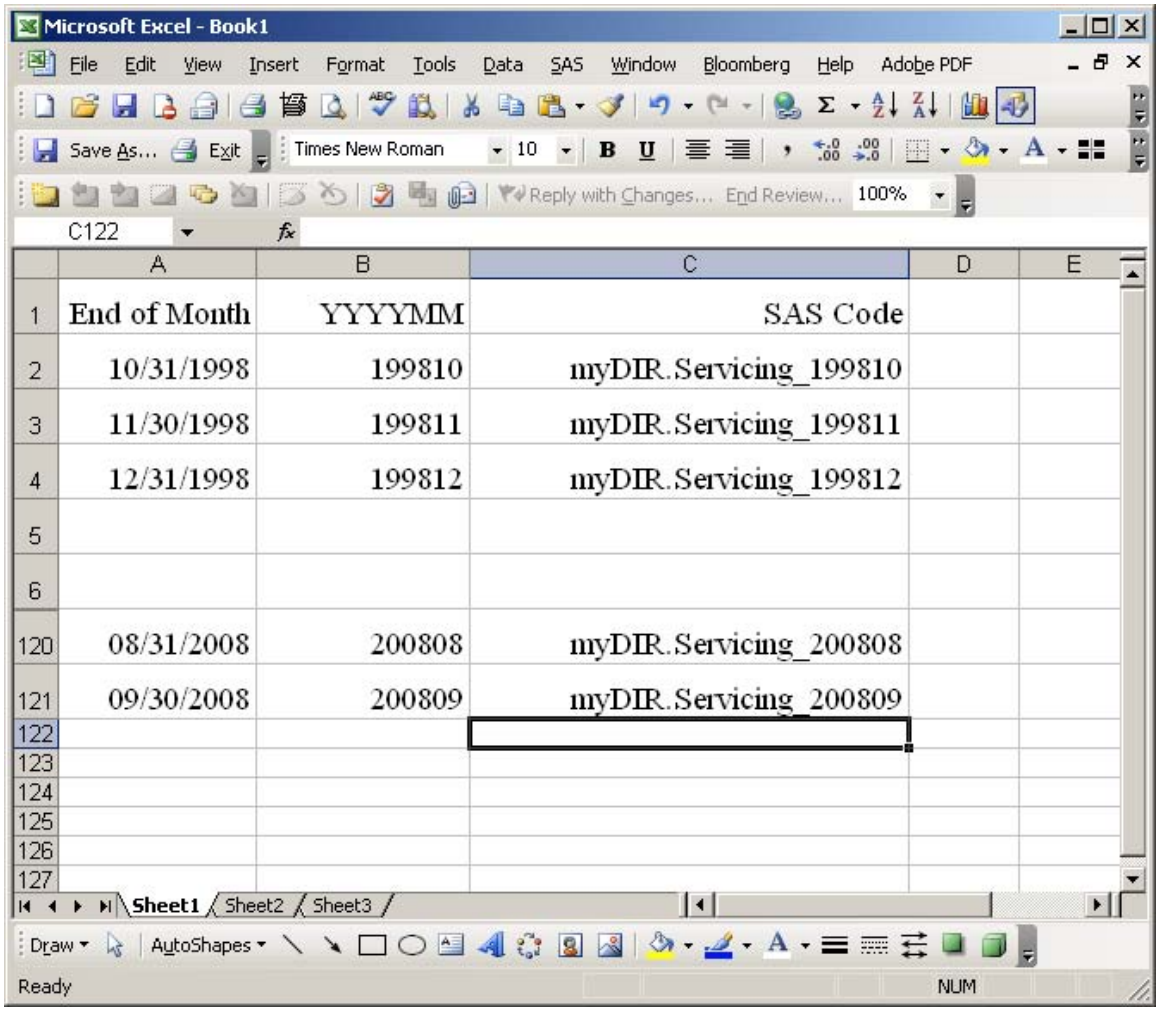
usage, the MaxBal, MaBalDate, and MaxMonOnBook will all be null. If PayDown is 1, the balance was built up and then paid off. It UseAfterPayDown is 1, the borrower has used the account, paid it off and then started to use the account again. These are just some examples to inspire readers to come up with some modifications to utilize this technique to extract date information out of variable names based the values of those variables.

Conclusions

Many datasets have either YYYYMM in their names or some variables containing YYYYMM. YYYYMM is not continuous, looping through them needs some work. We demonstrated techniques how to accomplish it with ease and a technique to extract date information embedded in variable names based on the value of those variables.

Appendix

Using Excel®to quickly build SAS code



	A	B	C	D	E
1	End of Month	YYYYMM	SAS Code		
2	10/31/1998	199810	myDIR.Servicing_199810		
3	11/30/1998	199811	myDIR.Servicing_199811		
4	12/31/1998	199812	myDIR.Servicing_199812		
5					
6					
120	08/31/2008	200808	myDIR.Servicing_200808		
121	09/30/2008	200809	myDIR.Servicing_200809		
122					
123					
124					
125					
126					
127					

078-2009

Step 1
Enter the starting date in cell A2. Enter the formula “=eomonth(A2, 1)” into cell A3 and copy it down to cell A121.

Step 2
To translate a date into YYYYMM string: enter the formula “=Text(A2, “YYYYMM”)” into cell B2 and copy it down to B121.

Step 3
Enter the formula “="myDIR.Servicing_" & B2” into cell C3 and copy it down to C121. Walla, you have one hundred lines of SAS code!