Paper 082-2010

Using the External File Name to create File Date variables

Peter Cerussi, Northrop Grumman, Monterey, CA

ABSTRACT:

Creating a file date for each input file, when one does not exist in the file, becomes a tedious task as the number of files grows. Different methods can be used, such as reading in all the files at once, using a file counter, the Job File Control Block (JFCB), and implementing macros. The key was to find a solution that reduces both the programmer's time and CPU time. When all methods were compared to reading in each file in its own data step, the file counter and JFCB both increased CPU time, but decreased the amount of hard coding necessary, while using a macro decreased the amount of hard coding and CPU time. This made the macro the most efficient solution when looking at both the programmer's time and CPU time. This method can be used if the file date is anywhere in the filename.

INTRODUCTION:

When the need arises to do analysis based on the file date of the file, it can be time consuming to hard code the file date in for each file. This is especially true when the analysis is needed covering multiple years on a file that is submitted every month. There are different techniques available to accomplish this task without having to hard code the file date. One of the techniques, the Job File Control Block, is specific to working on the mainframe. The others can be used on either a mainframe or PC. Since these programs were run on a mainframe, actual CPU time can vary slightly depending on other processes that were executing when the code was run. Unless otherwise stated, each method reads in monthly files from January 2008 – December 2008.

METHOD 1: READ IN EACH FILE IN A SEPARATE DATA STEP

To give a baseline for the CPU time, I hard coded the file date for one year's worth of monthly files. The example below shows the code for reading in a pay file for January 2008. "0801" in the Dsname (DSN) is the file date of the file in the file name.

```
OPTIONS ERROR=1 MERGENOBY=WARN;

DATA ONE;
INFILE ADY1;
INPUT
@109     CPS        $CHAR1.@;
IF CPS IN ('1','2');
INPUT
@550     COMP       $CHAR1.
@001     SVC          $CHAR1.
@038     SEX          $CHAR1.
@230     DIVING        PD3.
;

RETAIN FILEDATE 200801;

RUN;

NOTE: The infile ADY1 is:
      Dsname=ABCDEF.GHIJKL.MNOPQR.STUVWX.P0801,
      Unit=3390,Volume=SMS702,Disp=SHR,Blksize=27500,
      Lrecl=550,Recfm=FB,Creation=2008/01/31

NOTE: 1536612 records were read from the infile ADY1.
NOTE: The data set WORK.ONE has 1488161 observations and 6 variables.
NOTE: The DATA statement used 18.46 CPU seconds and 16980K.
```

The above code is repeated for the other 11 monthly files for 2008, and the total time needed to read in all 12 files was 237.43 CPU seconds. To get all 12 files into the same data set required creating a 13th data set with every observation from the first 12 data sets. To accomplish this efficiently, I used the append procedure, with the code below.

```
PROC APPEND BASE=APPENDED DATA=ONE;
PROC APPEND BASE=APPENDED DATA=TWO;
```

```
PROC APPEND BASE=APPENDED DATA=THREE;
PROC APPEND BASE=APPENDED DATA=FOUR;
PROC APPEND BASE=APPENDED DATA=FIVE;
PROC APPEND BASE=APPENDED DATA=SIX;
PROC APPEND BASE=APPENDED DATA=SEVEN;
PROC APPEND BASE=APPENDED DATA=EIGHT;
PROC APPEND BASE=APPENDED DATA=NINE;
PROC APPEND BASE=APPENDED DATA=TEN;
PROC APPEND BASE=APPENDED DATA=ELEVEN;
PROC APPEND BASE=APPENDED DATA=TWELVE;
```

The time needed to create the data set **APPENDED** was 3.37 CPU seconds, which brought the total time of this process to 240.80 CPU seconds. The frequency procedure on the variable **FILEDATE** was performed to show that every record has the correct file date. Reports would then be created, broken out by **FILEDATE**.

```
PROC FREQ DATA=ALL;
TABLE FILEDATE;

The FREQ Procedure
```

FILEDATE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0801	1488161	8.22	1488161	8.22
0802	1490326	8.23	2978487	16.46
0803	1490512	8.23	4468999	24.69
0804	1491834	8.24	5960833	32.93
0805	1489804	8.23	7450637	41.16
0806	1504852	8.31	8955489	49.48
0807	1516234	8.38	10471723	57.85
0808	1518691	8.39	11990414	66.24
0809	1522610	8.41	13513024	74.66
0810	1527256	8.44	15040280	83.09

The having the file date hardcoded became impractical when dealing with multiple years of monthly files. The next question is what technique will allow the programmer to set the **FILEDATE** variable equal to the file date at the end of the DSN.

METHOD 2: FILE COUNTER

The first technique covered used the **EOV** variable to create a file counter and **If-Then** statements to assign the proper file date to the record. In this example, **FNR** was set equal to the **EOV** variable. In the code below, the counter **FCNT** was increased by 1 every time a new file was encountered.

```
DATA FILECNT;
INFILE ADY1 EOV=FNR;
   /****  CREATES FILE DATE FIELD  ****/
IF _N_ = 1 THEN FNR=1;  /*****************************************/
IF FNR=1 THEN DO;       /* ONLY INCREASE FCNT ON THE FIRST OBS OF*/
  FCNT+1;               /* EACH FILE                             */
  FNR=0;                /*****************************************/
END;
INPUT
@109     CPS       $CHAR1.@;
IF CPS IN ('1','2');
INPUT
@550     COMP      $CHAR1.
@001     SVC       $CHAR1.
@038     SEX       $CHAR1.
@230     DIVING       PD3.
;

IF FCNT=1 THEN FILEDATE=200801;
ELSE IF FCNT=2  THEN FILEDATE=200802;
ELSE IF FCNT=3  THEN FILEDATE=200803;
ELSE IF FCNT=4  THEN FILEDATE=200804;
ELSE IF FCNT=5  THEN FILEDATE=200805;
```

```
ELSE IF FCNT=6  THEN FILEDATE=200806;
ELSE IF FCNT=7  THEN FILEDATE=200807;
ELSE IF FCNT=8  THEN FILEDATE=200808;
ELSE IF FCNT=9  THEN FILEDATE=200809;
ELSE IF FCNT=10 THEN FILEDATE=200810;
ELSE IF FCNT=11 THEN FILEDATE=200811;
ELSE IF FCNT=12 THEN FILEDATE=200812;
```

The code produced the below output.

```
NOTE: The data set WORK.FILECNT has 18100230 observations and 7
      variables.
NOTE: The DATA statement used 270.01 CPU seconds and 16980K.
```

This technique required the programmer to know the order that the external files were being read in to ensure the correct file date was associated with each record. This technique used more time than reading in each file in its own SAS data step, but required less hard coding. However, it also introduced a possible source of programmer error when **FILEDATE** was created based on the value of **FCNT**.

```
PROC FREQ DATA=ALL;
TABLE FILEDATE;

The FREQ ProcedurE
```

FILEDATE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0801	1488161	8.22	1488161	8.22
0802	1490326	8.23	2978487	16.46
0803	1490512	8.23	4468999	24.69
0804	1491834	8.24	5960833	32.93
0805	1489804	8.23	7450637	41.16
0806	1504852	8.31	8955489	49.48
0807	1516234	8.38	10471723	57.85
0808	1518691	8.39	11990414	66.24
0809	1522610	8.41	13513024	74.66
0810	1527256	8.44	15040280	83.09

METHOD 3: JFCB AND FILE COUNTER

The JFCB contains information on the DSN when the file is read in with mainframe SAS. The JFCB is 176 bytes long, the first 44 of which contain the DSN. JFCB is an option which needs to be specified on the infile statement. The JFCB was used, in conjunction with the file counter, to allow the programmer to bypass using a series of **If-Then** statements by reading the file date from the DSN. Each time SAS read in a new file, a scan function was used on the JFCB to set the variable **FILEDATE** equal to the file date in the filename.

```
DATA ADY;
INFILE ADY1 EOV=FNR JFCB=JFCB;
IF _N_ = 1 THEN FNR=1;            /*****************************************/
IF FNR=1 THEN DO;                 /*ONLY SCAN JFCB ON THE FIRST OBS OF EACH*/
  FILEDATE = SCAN(SUBSTR(JFCB,1,44),-1,'P');/*FILE TO INCREASE EFFICIENCY  */
  FCNT+1                                  /*********************************/
  FNR=0;
END;
ELSE IF _N_ ¬=1 THEN DO; /*****************************************/
   RETAIN FILEDATE;      /* RETAIN FILEDATE FROM FIRST OBS OF FILE*/
END;                     /*****************************************/
INPUT
@109     CPS       $CHAR1.@;
IF CPS IN ('1','2');
INPUT
@550     COMP         $CHAR1.
@001     SVC          $CHAR1.
@038     SEX          $CHAR1.
@230     DIVING          PD3.
;
```

In this example the file date is contained at the end of the DSN, ABCDEF.GHIJKL.MNOPQR.STUVWX.P0801, after the 'P'. The first 44 bytes of the JFCB were scanned from right to left, and set **FILEDATE** to all the characters scanned until 'P'.

```
NOTE: The data set WORK.ADY has 18100230 observations and 7 variables.
NOTE: The DATA statement used 270.70 CPU seconds and 17057K.
```

This method did not increase CPU time when compared to the file counter with **If-Then** statements. However, this method still used about 25 more seconds of CPU time compared to reading in each file in its own data set. If the job required reading in multiple years, the difference in the two methods would become more apparent.

```
PROC FREQ DATA=ALL;
TABLE FILEDATE;

The FREQ Procedure
```

FILEDATE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0801	1488161	8.22	1488161	8.22
0802	1490326	8.23	2978487	16.46
0803	1490512	8.23	4468999	24.69
0804	1491834	8.24	5960833	32.93
0805	1489804	8.23	7450637	41.16
0806	1504852	8.31	8955489	49.48
0807	1516234	8.38	10471723	57.85
0808	1518691	8.39	11990414	66.24
0809	1522610	8.41	13513024	74.66
0810	1527256	8.44	15040280	83.09

METHOD 4: JFCB AND FILE COUNTER IN A MACRO

The first step was to create macro variables for each of the 12 monthly files per year. That was accomplished using the **CALL SYMPUT** function. Then use **%IF-%THEN** to account for the fact not all files exist for the current year. For example, files for 2010 only exist through February at the time this paper was written. If someone wanted to run this macro for files in 2010, it would not try to import a file from March 2010 that does not exist. After the macro variables have been created, the code from Method 2 is executed.

```
%MACRO RSFM(DATE=);

 DATA _NULL_;
    CALL SYMPUT('DATE01',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'01')));
    CALL SYMPUT('DATE02',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'02')));
    CALL SYMPUT('DATE03',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'03')));
    CALL SYMPUT('DATE04',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'04')));
    CALL SYMPUT('DATE05',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'05')));
    CALL SYMPUT('DATE06',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'06')));
    CALL SYMPUT('DATE07',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'07')));
    CALL SYMPUT('DATE08',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'08')));
    CALL SYMPUT('DATE09',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'09')));
    CALL SYMPUT('DATE10',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'10')));
    CALL SYMPUT('DATE11',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'11')));
    CALL SYMPUT('DATE12',
               (COMPRESS('ABCDEF.GHIJKL.MNOPQR.STUVWX.P0'||&DATE||'12')));

 RUN;
```

```
    %IF &DATE=10 %THEN %DO;
       FILENAME RSF ("&DATE01", "&DATE02");

    %END;
    %ELSE %DO;
FILENAME RSF
    ("&DATE01","&DATE02","&DATE03","&DATE04","&DATE05","&DATE06",&DATE07",
     "&DATE08","&DATE09","&DATE10","&DATE11","&DATE12");
    %END;

    DATA ADYMACRO;
    INFILE RSF EOV=FNR JFCB=JFCB;
    IF _N_ = 1 THEN FNR=1;              /*****************************************/
    IF FNR=1 THEN DO;                    /* ONLY SCAN JFCB ON THE FIRST OBS OF EACH*/
    FILEDATE = SCAN(SUBSTR(JFCB,1,44),-1,'P');/*  FILE TO INCREASE EFFICIENCY    */
      FNR=0;                                  /***********************************/
    END;
    ELSE IF _N_ ¬=1 THEN DO;  /*****************************************/
       RETAIN FILEDATE;       /* RETAIN FILEDATE FROM FIRST OBS OF FILE*/
    END;                      /*****************************************/
    INPUT
    @109     CPS       $CHAR1.@;
    IF CPS IN ('1','2');
    INPUT
    @550     COMP      $CHAR1.
    @001     SVC          $CHAR1.
    @038     SEX          $CHAR1.
    @230     DIVING           PD3.
    ;

    PROC FREQ DATA=ADYMACRO;
    TABLE FILEDATE;

    %MEND;

    %RSFM(DATE=8);
```

The log output from the above code:

```
The data set WORK.ADYMACRO has 18100230 observations and 6 variables.
The DATA statement used 273.59 CPU seconds and 17583K.
```

The macro **RSFM** took 273.59 seconds to read in all 12 monthly files for 2008. This takes almost the same amount of time as method 3, the advantage this can be repeated for any desired year without having to list each filename individually.

```
PROC FREQ DATA=ALL;
TABLE FILEDATE;

The FREQ Procedure
```

FILEDATE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0801	1488161	8.22	1488161	8.22
0802	1490326	8.23	2978487	16.46
0803	1490512	8.23	4468999	24.69
0804	1491834	8.24	5960833	32.93
0805	1489804	8.23	7450637	41.16
0806	1504852	8.31	8955489	49.48
0807	1516234	8.38	10471723	57.85
0808	1518691	8.39	11990414	66.24
0809	1522610	8.41	13513024	74.66
0810	1527256	8.44	15040280	83.09

METHOD 5: USING AN ARRAY OF MACRO VARIABLES IN THE FILENAME

In this method, a macro is created to read in all the months for a calendar year. This is done by setting the Macro Variable **Year** to the 2 byte year you want read in, and then using a %DO loop to control which month is being read in. The Macro Variable **Months** is defaulted to twelve since only the current calendar year does not have 12 months. To avoid reading in non existent months, **Months** can be declared as the current month.

To create an array of macro variables to control for the current month, macro variables are created for each variable in the array. Think of **MONTH** being the name of the array while the number corresponds to the specific macro variable in the array. For example, **MONTH1** would be the first macro variable in the array and **MONTH2** would be the second macro variable in the array.

A %DO loop is used to go through the macro variables, setting the data set name, filename, and file date to the current year and month. Each monthly file is read into a different data set because that was shown to be the most efficient of all the above methods, and then appended to a data set which will contain all the records from all the files read in.

```
OPTIONS ERRORS=1 MISSING=0 MPRINT MERGENOBY=WARN;

%MACRO FILEDATES(YEAR=, MONTHS=12);
%LET MONTH1=01;
%LET MONTH2=02;
%LET MONTH3=03;
%LET MONTH4=04;
%LET MONTH5=05;
%LET MONTH6=06;
%LET MONTH7=07;
%LET MONTH8=08;
%LET MONTH9=09;
 %LET MONTH10=10;
 %LET MONTH11=11;
 %LET MONTH12=12;

 %DO I=1 %TO &MONTHS;
 DATA ADY&&MONTH&I;
 INFILE "ABCDEF.GHIJKL.MNOPQR.STUVWX.P&&YEAR&&MONTH&I";
 INPUT
 @109     CPS        $CHAR1.@;
 IF CPS IN ('1','2');
 INPUT
 @550     COMP        $CHAR1.
 @001     SVC           $CHAR1.
 @038     SEX           $CHAR1.
 @230     DIVING             PD3.
 ;

 IF _N_=1 THEN DO;
    FILEDATE="&&YEAR&&MONTH&I";
    RETAIN FILEDATE;
 END;

 PROC APPEND BASE=DATASET DATA=ADY&&MONTH&I;
 RUN;

 %END;
 %MEND;

 %FILEDATES(YEAR=08);
```

This code produces the following output in the log window.

```
NOTE: The infile "ABCDEF.GHIJKL.MNOPQR.STUVWX.P0801" is:
      Dsname=ABCDEF.GHIJKL.MNOPQR.STUVWX.P0801,
      Unit=3390,Volume=SMSB65,Disp=SHR,Blksize=27500,
      Lrecl=550,Recfm=FB,Creation=2008/01/31

NOTE: 1536612 records were read from the infile
      "ABCDEF.GHIJKL.MNOPQR.STUVWX.P0801".
NOTE: The data set WORK.ADY01 has 1488161 observations and 6 variables.
```

```
NOTE: The DATA statement used 20.14 CPU seconds and 17236K.

NOTE: The address space has used a maximum of 628K below the line and
      18324K above the line

MPRINT(FILEDATES):   PROC APPEND BASE=DATASET DATA=ADY01;
MPRINT(FILEDATES):   RUN;

NOTE: Appending WORK.ADY01 to WORK.DATASET.
NOTE: BASE data set does not exist. DATA file is being copied to BASE
      file.
NOTE: There were 1488161 observations read from the data set WORK.ADY01.
NOTE: The data set WORK.DATASET has 1488161 observations and 6 variables.
NOTE: The PROCEDURE APPEND used 0.50 CPU seconds and 21008K.
```

This was repeated for all 12 months of 2008. The total time it took to read in all the files was 230.46 CPU seconds, and the time to complete all the Proc Append procedures was 2.61 CPU seconds, which brought the total time to 233.07 CPU seconds. This is close to 10 seconds less CPU time compared to method 1, and did not involve having to hard code the file date and Proc Append. A Proc Freq on the data set **DATASET** shows it has the same record count for each file date.

```
PROC FREQ DATA=ALL;
TABLE FILEDATE;

The FREQ Procedure
```

FILEDATE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0801	1488161	8.22	1488161	8.22
0802	1490326	8.23	2978487	16.46
0803	1490512	8.23	4468999	24.69
0804	1491834	8.24	5960833	32.93
0805	1489804	8.23	7450637	41.16
0806	1504852	8.31	8955489	49.48
0807	1516234	8.38	10471723	57.85
0808	1518691	8.39	11990414	66.24
0809	1522610	8.41	13513024	74.66
0810	1527256	8.44	15040280	83.09

The macro above can only handle reporting for a calendar year, which would present a problem when a report is needed for a fiscal year. Adding conditional **%If %Then %Do** logic gave the macro flexibility to read in up to 12 consecutive months over 2 calendar years. There were a couple other changes made to reduce to coding. The DSN before any macro variables in the infile statement was changed into a parameter of the macro. Also, a second macro **%CODE** was created to control the input statement, which was nested inside the macro **%FYDATES**.

```
/*********************************************************************/
/* THIS MACRO IS THE CODE FOR THE DATA STEP IN THE BELOW MACRO.       */
/* CONTROLS INPUT STATMENT AND OTHER DATA MANIPULATION THAT NEEDS     */
/* TO TAKE PLACE IN THE DATA STEP.  FILE DATE NOT CREATED IN THIS     */
/* MACRO                                                             */
/*********************************************************************/

%MACRO CODE();
   INPUT
   @109     CPS       $CHAR1.@;
   IF CPS IN ('1','2');
   INPUT
   @550     COMP      $CHAR1.
   @001     SVC          $CHAR1.
   @038     SEX          $CHAR1.
   @230     DIVING           PD3.
   ;
%MEND;
```

The parameters of the macro %FYDATES are as follows:
YEAR – The year the fiscal year ends (last 2 bytes of the year)

DSN – The portion of the filename preceding the file date
LASTMONTH – The last calendar month being read in
FYSTART – The first calendar month being read in

```
OPTIONS ERRORS=1 MISSING=0 MPRINT MLOGIC SYMBOLGEN MERGENOBY=WARN;

%MACRO FYDATES(YEAR=, DSN=, LASTMONTH=9, FYSTART=10);

%LET MONTH1=01;
%LET MONTH2=02;
%LET MONTH3=03;
%LET MONTH4=04;
%LET MONTH5=05;
%LET MONTH6=06;
%LET MONTH7=07;
%LET MONTH8=08;
%LET MONTH9=09;
%LET MONTH10=10;
%LET MONTH11=11;
%LET MONTH12=12;
/*********************************************************************/
/* DATE STEP: SETS MACRO VARIABLE &YEAR1 EQUAL TO THE CALENDAR YEAR    */
/*            OF THE FIRST MONTH BEING READ IN.  THIS ACCOMODATES      */
/*            FOR 12 MONTHS WHICH SPANS 2 CALENDAR YEARS.              */
/*                                                                     */
/*            SETS MARCO VARIBALE &YEAR2 TO CALENDAR YEAR OF THE       */
/*            LAST MONTH BEING READ IN                                 */
/*********************************************************************/
DATA _NULL_;
%IF &YEAR>10 %THEN %DO;
    YEAR1 = &YEAR-1;
    CALL SYMPUT('YEAR1', YEAR1);
    %LET YEAR2 = &YEAR;
%END;
%IF &YEAR=00 %THEN %DO;
    %LET YEAR1 = 99;
    %LET YEAR2=00;
%END;
%IF (&YEAR<=10 AND &YEAR¬=00) %THEN %DO;
    YEAR1 = &YEAR-1;
    CALL SYMPUT('YEAR1','0'||TRIM(LEFT(PUT(YEAR1,1.))));
    %LET YEAR2 = &YEAR;
%END;

/*********************************************************************/
/* %IF STATEMENT: EXECUTES THE FOLLOWING CODE IF THE MONTHS BEING      */
/*                READ IN FALL WITHIN ONE CALENDAR YEAR                */
/*********************************************************************/
%IF &LASTMONTH>=&FYSTART %THEN %DO;

/*********************************************************************/
/* %DO STATEMENT: CYCLES FROM MONTH[FYSTART] TO MONTH[LASTMONTH]       */
/*                WHEN SETTING DATA SET NAME AND INFILE STATMENT       */
/*                AND IN INFILE STATEMENT                              */
/*********************************************************************/
   %DO I=&FYSTART %TO &LASTMONTH;
   DATA ADY&&MONTH&I;
   INFILE "&DSN&YEAR&&MONTH&I";
   %CODE();

   IF _N_=1 THEN DO;
      FILEDATE="&YEAR&&MONTH&I";
      RETAIN FILEDATE;
   END;

   PROC APPEND BASE=FY&YEAR DATA=ADY&&MONTH&I;
   RUN;
```

```
   %END;
%END;

/*****************************************************************/
/* IF STATEMENT: EXECUTES THE FOLLOWING CODE WHEN THE MONTHS     */
/*               SPAN 2 CALENDAR YEARS                           */
/*****************************************************************/
%IF &LASTMONTH<&FYSTART %THEN %DO;

 /*******************************************************************/
 /* DO STATEMENT: J SETS &&YEAR&J TO EITHER YEAR1 OR YEAR2 DEPENDING */
 /*               ON THE VALUE OF J                                 */
 /*******************************************************************/
   %DO J=1 %TO 2;
      %IF &J=1 %THEN %DO;

/*********************************************************************/
/* DO STATEMENT: I SETS &&MONTH&I TO MONTH1 TO MONTH12 DEPENDING ON  */
/*               THE VALUE OF I                                      */
/*********************************************************************/
         %DO I=&FYSTART %TO 12;
         DATA ADY&&MONTH&I;
         INFILE "&DSN&&YEAR&J&&MONTH&I";
         INPUT
         %CODE();

         IF _N_=1 THEN DO;
            FILEDATE="&&YEAR&J&&MONTH&I";
            RETAIN FILEDATE;
         END;

         PROC APPEND BASE=FY&YEAR DATA=ADY&&MONTH&I;
         RUN;

         %END;
      %END;
      %IF &J=2 %THEN %DO;

/*********************************************************************/
/* DO STATEMENT: I SETS &&MONTH&I TO MONTH1 TO MONTH12 DEPENDING ON  */
/*               THE VALUE OF I                                      */
/*********************************************************************/
         %DO I=1 %TO &LASTMONTH;
         DATA ADY&&MONTH&I;
         INFILE "&DSN&&YEAR&J&&MONTH&I";
         INPUT
         %CODE();

         IF _N_=1 THEN DO;
            FILEDATE="&&YEAR&J&&MONTH&I";
            RETAIN FILEDATE;
         END;

         PROC APPEND BASE=FY&YEAR DATA=ADY&&MONTH&I;
         RUN;

         %END;
      %END;
   %END;
%END;

PROC FREQ DATA=FY&YEAR;
TABLE FILEDATE/MISSING LIST;
RUN;
%MEND;
```

```
%FYDATES(YEAR=08,DSN=ABCDEF.GHIJKL.MNOPQR.STUVWX.P,FYSTART= 1,
  LASTMONTH=12);
```

This macro was executed for the calendar year 2008 to be able to compare the differences in time. The following is the log output for the first month.

```
NOTE: 1536612 records were read from the infile
"ABCDEF.GHIJKL.MNOPQR.STUVWX.P0801".
NOTE: The data set WORK.ADY01 has 1488161 observations and 6 variables.
NOTE: The DATA statement used 20.54 CPU seconds and 17318K.

NOTE: Appending WORK.ADY01 to WORK.FY08.
NOTE: BASE data set does not exist. DATA file is being copied to BASE
file.
NOTE: There were 1488161 observations read from the data set WORK.ADY01.
NOTE: The data set WORK.FY08 has 1488161 observations and 6 variables.
NOTE: The PROCEDURE APPEND used 0.46 CPU seconds and 21089K.

The FREQ Procedure
```

FILEDATE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0801	1488161	8.22	1488161	8.22
0802	1490326	8.23	2978487	16.46
0803	1490512	8.23	4468999	24.69
0804	1491834	8.24	5960833	32.93
0805	1489804	8.23	7450637	41.16
0806	1504852	8.31	8955489	49.48
0807	1516234	8.38	10471723	57.85
0808	1518691	8.39	11990414	66.24
0809	1522610	8.41	13513024	74.66
0810	1527256	8.44	15040280	83.09

The program took 242.02 CPU seconds to read in all 12 files. The Proc Append procedure took 2.7 CPU seconds to put all observations into **WORK.FY08**, which brought the total time to 244.72 CPU seconds. This took only slightly more time than the previous macro, but has the advantage of being able to read 12 months over 2 years. It is faster than methods 1 through 4, and needs less coding.

The above macro was also run for fiscal year 2009, which goes from October 2008-September 2009. The macro did not lose any efficiency when spanning 2 calendar years, having read in and appended all records in 231.42 CPU seconds.

```
%FYDATES(YEAR=09,DSN=ABCDEF.GHIJKL.MNOPQR.STUVWX.P);

NOTE: The infile "ABCDEF.GHIJKL.MNOPQR.STUVWX.P0810" is:
      Dsname=ABCDEF.GHIJKL.MNOPQR.STUVWX.P0810,
      Unit=3390,Volume=SMS753,Disp=SHR,Blksize=27500,
      Lrecl=550,Recfm=FB,Creation=2008/10/29

NOTE: 1579315 records were read from the infile
      "ABCDEF.GHIJKL.MNOPQR.STUVWX.P0810".
NOTE: The data set WORK.ADY10 has 1527256 observations and 6 variables.
NOTE: The DATA statement used 15.80 CPU seconds and 17318K.

NOTE: Appending WORK.ADY10 to WORK.FY09.
NOTE: BASE data set does not exist. DATA file is being copied to BASE
      file.
NOTE: There were 1527256 observations read from the data set WORK.ADY10.
NOTE: The data set WORK.FY09 has 1527256 observations and 6 variables.
NOTE: The PROCEDURE APPEND used 0.47 CPU seconds and 21089K.

NOTE: There were 18484834 observations read from the data set WORK.FY09.
NOTE: The PROCEDURE FREQ printed page 1.
NOTE: The PROCEDURE FREQ used 38.30 CPU seconds and 22957K.

The SAS session used 269.72 CPU seconds and 22957K.
```

```
The FREQ Procedure

                                                 Cumulative    Cumulative
 FILEDATE            Frequency     Percent      Frequency      Percent
 ---------------------------------------------------------------------
 0801                  1488161        8.22        1488161         8.22
 0802                  1490326        8.23        2978487        16.46
 0803                  1490512        8.23        4468999        24.69
 0804                  1491834        8.24        5960833        32.93
 0805                  1489804        8.23        7450637        41.16
 0806                  1504852        8.31        8955489        49.48
 0807                  1516234        8.38       10471723        57.85
 0808                  1518691        8.39       11990414        66.24
 0809                  1522610        8.41       13513024        74.66
 0810                  1527256        8.44       15040280        83.09
```

CONCLUSION:

A table has been provided below to compare the differences in CPU time each method took.

Method	CPU Time (seconds)
1) Read in Each File in a Different Data Step	240.80
2) File Counter	270.01
3) JFCB & File Counter	270.70
4) JFCB & File Counter in Macro	273.59
5) Array of Marco Variables	244.72

A macro is the most efficient way to read in a years worth of data, when looking at both CPU time and programming time. If a report needs to be done for the last 10 fiscal years, it would be infeasible to hard code the date of 144 monthly files into the program. To do so would increase the opportunity for programming error while providing no benefits in CPU time.

REFERENCES:

1. Carr, David W. When Proc Append Makes More Sense Than the DATA STEP. SGF 2008, San Antonio, TX. (http://www2.sas.com/proceedings/forum2008/085-2008.pdf)
2. Stanley, Don. Using JFCB. (http://www.angelfire.com/wizard/don_stanley/sas_tips/jfcb.htm)
3. Philip, Stephan. SAS Macro: Beyond the Basics. SGF 2008, San Antonio, TX. (http://www2.sas.com/proceedings/forum2008/045-2008.pdf)

CONTACT INFORMATION

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

Peter Cerussi
Email: Peter.Cerussi@gmail.com