

Usage of SYSDATE for Dynamic and Internal Updates during the File Maintenance Run in the Macro-Based Information System

Jeff F. Sun, Blue Cross Blue Shield of North Carolina, Chapel Hill, NC

Abstract

The file maintenance for a large macro-based information system is a time-consuming and complex process which generally involves updating a lot of programs for each run. These updates should be logically and systematically organized so that the whole system can be smoothly and timely updated. Since most file maintenance runs are periodic and routine processes, the updates to these programs can be done internally and dynamically by the host system using the system information to achieve efficiency and accuracy. This paper provides a general approach on how to use the automatic macro variable SYSDATE in the internal and dynamic updates, i.e. file names, time periods, or texts, in the macro-based information system during the file maintenance run. The logical design, information retrieval, macro coding, installation, and validation checking will be addressed in detail.

I. Introduction

The file maintenance for a large macro-based information system can be a tedious and labor-intensive process if the production system has many manual interventions. This process generally involves updating a lot of programs for each run. So accuracy, consistency, and efficiency of updates are essential in a large production system maintenance run. These updates should be logically and systematically organized so that the whole system can be smoothly and timely updated.

There are two ways we can use to update the routine maintenance run internally and dynamically. One is to retrieve needed information directly from the data files. The other way may be more accurate and safer, that is, getting information from the SAS environment. Existing document does a good job at describing the practice of deriving information from the data file (for example, Knoop, 1993); however, it does not provide the approaches of how to retrieve needed information from the SAS environment. Since most file maintenance runs are periodic and routine processes, the updates to these programs can be done internally and dynamically by the host system using the system information to achieve efficiency and accuracy. In other words, we can make the information system more efficient, accurate, and consistent by deriving as much information from the SAS environment as possible. The use of this system can result in greater programming and maintaining efficiency, accuracy, and consistency.

This paper provides a general approach on how to use the automatic macro variable SYSDATE to build utility macros in the internal and dynamic updates, i.e. file names, time periods, or texts, in a macro-based information system during a file maintenance run. Section II discusses the major features of a routine maintenance run of a macro-based information system. In the next section the types of updates in a macro-based information system are addressed in detail. Section IV presents the logical design and physical development of this kind of utility macros, including information retrieval, macro code, installation, and

validation. A case study is given in Section V to provide some insight into how to design, develop, and code this kind of utility macros. Conclusions are presented in the final section.

II. A Routine Maintenance Run

The monthly data processing and reporting is probably the most common type of business practice. Every month the SAS system updates the information system by updating the particular information in the production programs, for example, the time periods, file names, texts, and other information. The monthly transaction files are processed. The resulting files could be used to produce reports or update the master files.

For a large information system, these updates may be very large and time-consuming. Even though some macro techniques are used in this process to reduce the manual work, for example, global updates to macro variables¹, the manual updates are still time-consuming and error prone if there are still a lot of manual interventions, for example, manual updates to values of macro variables, in the maintenance run.

Here are a few problems that could arise if there are many manual updates, for example, inaccuracy in macro value specification, or labor-intensive key-in in the manual updates, etc. This requires an advanced technique to use the macro facility to create functional macros which can be used to internally and dynamically update the information system. In other words, once each production program is submitted, the requested information will be updated during the macro compilation time.

By using the SAS macro facility, a number of utility macros can be developed to aid system

¹ Most common and easiest macro technique used is the global updates to macro variables. This is an important data management feature which provides us an ability to update a variable in one place, and "cascade" the update to all related data sets, DATA steps, or PROC's. (Michel, 1993).

operators to perform routine maintenance tasks. Having these utility macros is a proven time saver that it has been seen to improve system operators productivity dramatically. This paper presents some useful techniques that can be applied to the design, definition, and implementation of dynamic and internal updates.

III. Updates in A Maintenance Run

There are many possible updates in a macro-based information system. It depends on the programmers' programming style and information availability in a system. The following lists several common updates in a production system.

a. Time Periods

The core element in the time periods is the "time point" which represents a particular time point. It could be the year, the month, or the day. There are many formats in practice.

For example, the calendar year value of 1997 can be represented as 1997, or 97, or the year of any SAS date value between 13515 and 13879, or the year of any Gregorian date value between 01JAN97 and 31DEC97, or the year of any Julian date value between 97001 and 97365. The calendar month value of September 1997 can be represented as September, or September 1997, or the month of any Gregorian date value between 01SEP97 and 30SEP97, or the month of any SAS date value between 13758 and 13787, or the month of any Julian date value between 97243 and 97274. For the calendar date value of September 22, 1997 we can have 092297, or 9/22/97, or 22SEP97, or 972209, or 22SEP1997, or September 22 1997, or 97265, or 13779, etc.

Some common examples of date or time period used in a SAS program can be described as follows.

- IF BEGDATE GE '01JAN97'D;
- WHERE 97001 GE CSDATE GE 97365;

- LENGTH=BEGDATE - 13787;
- IF NEWDATE EQ '22SEP97';

The above are just some examples of date used in the SAS program. In practice it is much more complex.

b. Texts

The text updates in a macro-based information system include texts for titles, footnotes, notes, legends, or labels for axis's, etc. We can use %LET statement, or SYMPUT routine to create macro variables. It is commonly used in the following places.

- in the TITLE statement:

```
TITLE2 "Table 1 &MYTITLE";
```

- in the FOOTNOTE statement:

```
FOOTNOTE2 "Note: &MYNOTE";
```

- in the AXIS statement:

```
AXIS2 LABEL=(a=-90 r=90 "&MYTITLE");
```

- in the NOTE statement:

```
NOTE h=6 pct f=swissl "&MYNOTE";
```

- in the LEGEND statement:

```
LEGEND1 LABEL=(h=4 pct f=swissl "&MYLEG");
```

There are many other text updates in a SAS program. We are not going to list all of them in this paper.

c. Initialization and Assignment

A macro variable is useful for storing the initial values we need to assign to the array elements, for example,

```
ARRAY MLIST(12) _TEMPORARY_ (&LIST);
```

or assign a value to a particular variable:

```
WHERE REGION=&REG;
```

d. File Names

The file name for the production information system is characterized by a file name convention; for example, the monthly file maintenance often follows a name convention, including *Month*, *Year*, and *File name*. For example,

```
USERNAME.MAINDIR.SUBDIR.FILENAME
```

The main directory can be a project name or a group name. The sub-directory could be the month when the files are created. For example, in the PC environment, the file names are as follows.

```
c:\finance\project1\sep97\sales.sd2
c:\finance\project1\sep97\costs.sd2
```

in the MVS environment, the files may have the following names:

```
ABCCOMP.FINANCE.PROJECT1.SEP97.SALES
ABCCOMP.FINANCE.PROJECT1.SEP97.COSTS
```

Besides the above specifications, there are many other manual updates required in the production system. This is product or system dependent. All this information can be internally and dynamically updated by the host system itself with utility macros installed. Of course we have to develop these macro utility ourselves. This is the main focus of the next section.

IV. Design and Development

Logical Design

When a person updates the values of macro variables in a production system, he must first identify the time when the system runs and pick up the information accordingly. This process can be programmatically accomplished. Instead of manually changing the values of these macro variables whenever their contents change, these macro variables are instead dynamically and internally updated by the utility macros.

The logic of building a dynamic and internal utility macro can be illustrated using the Figure 1.

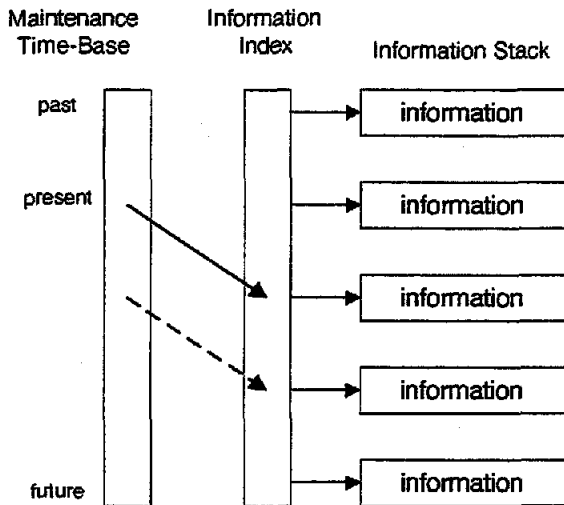


Figure 1. Logic of Building a Utility Macro

The needed information is logically, maybe not physically, organized by the information index, which is a time index. This information may not physically exist, for instance, in a SAS data file. If there were a physical file for this information, it could have had the format shown in Table 1 below.

Table 1. An Imagine Update Information Stack

Index	Update Information Stack
...	...
MAR97	March 1997, MAR, 1997, 97, 3, 01JAN97, 30JUN97, 123, 1234, 16, ...
APR97	April 1997, APR, 1997, 97, 4, 01FEN97, 31JUL97, 345, 567, 24, ...
MAY97	May 1997, May, 1997, 97, 5, 01MAR97, 31AUG97, 456, 986, 34, ...
JUN97	June 1997, June, 1997, 97, 6, 01APR97, 30SEP97, 629, 347, 45, ...
...	...

When an operator manually updates this information, he needs to identify the maintenance time and make an adjustment to the time base of information index to obtain the needed

information from the information stack. However, the process of identification and justification may be time-consuming and run into errors easily if there are many programs in the system and many update requirements in each program. With the utility macros, this process can be programmatically done by computer. The algorithm of this updating process is to identify the present year and month, then adjust the time point to the needed information index, and obtain the information based on this information index.

The approach used here is very similar to the table look-up technique. Table lookup is a way to add a variable to a data step. In our updating process, we create the 'key variable', which is a time point, dynamically from the system information, search for the 'lookup variable', which is the index, from our design tables (i.e. a logically organized information stack) for lists, organize values of 'lookup variables', and change the values of the macro variables.

The signal we use to update the information system is the time when we execute the system. Each time when we run the system, we implicitly change the values for all macro variables. The new values represent new information in the system. This is the algorithm of the dynamic and internal updating process for the production system with SAS macro facility.

Physical Development

The physical development of a utility macro for a production system consists of the following three steps:

- Step 1. Retrieve the dynamic information from the host system;
- Step 2. Extract the needed information from the information stack based on the information index;
- Step 3. Organize the requested information and represent it internally;

This paper uses a monthly run production system as an example to illustrate the idea of design and

development of utility macros. Since in a routine maintenance run most required changes are standardized, the maintenance time information can be extracted from the SAS environment and used as a base point to track these updates.

Step1. Retrieve the Dynamic Information

When you invoke the SAS system, the macro processor creates automatic macro variables that supply a variety of information. Referencing automatic macro variables is exactly the same as you do macro variables that you create. One of the most common used automatic macro variables is SYSDATE.² The common usage of this automatic variable is to include today's date in a FOOTNOTE or NOTE statement. Besides this, SYSDATE can play a very useful role in the dynamic updates of the routine maintenance run for the macro-based information system.

The SYSDATE is an automatic macro variables created by the macro processor and it supplies information on the most current date. Since the Read/Write status for SYSDATE is 'read only', it guarantees that we can obtain the same information as that of system. This automatic variable is global and is created when you invoke the SAS system. By definition, it contains the date that a SAS job or session began executing. This auto variable supplies baseline information for a production maintenance. The basic format of SYSDATE is *ddmmmyy*, for example, 23SEP97.

The baseline information for a monthly-maintained production system is the month and year information, that is, the characters from position 3 to position 5 for month, and from position 6 to 7 for year.³ In this example, the month information is:

```
%SUBSTR(22SEP97,3,3) = SEP
```

² Using %SYSFUNC(TODAY(),DATE8.) can get the same date value.

³ We only discuss two-digit years in this paper. For the cases of four-digit years, please see the SUGI paper written by Langstone (1997).

```
%SUBSTR(22SEP97,6,2) = 97
```

Table 2 describes three different time categories based on the values of %SUBSTR(&SYSDATE,3,3) or %SUBSTR(&SYSDATE,6,2). This gives us a standard point to start with. A further conversion of this character to other formats will be very convenient.

Table 2. Three Different Time Categories

Category	Description	Examples
Past	LT %SUBSTR(&SYSDATE,6,2) LT %SUBSTR(&SYSDATE,3,3)	year: 96, 95 month: JAN, MAR
Present	EQ %SUBSTR(&SYSDATE,6,2) EQ %SUBSTR(&SYSDATE,3,3)	year: 97 month: SEP
Future	GT %SUBSTR(&SYSDATE,6,2) GT %SUBSTR(&SYSDATE,3,3)	year: 98, 99 month: OCT, NOV

The "list approach" is a good design choice for determining the index where the size of the list can be fixed, i.e. 12 months in a year, or seven days in a week. The strength of the "list approach" is that it is easy to understand and code. It is also convenient to return the needed information later on. The weakness of the list is that the elements in a list must be hardcoded into the program. So this approach is suitable for the case which has small number of elements in a list. Since the monthly production and reporting is the most common business practice, it seems like a nice touch to use the list approach which retrieves and returns information easily.

```
%MACRO CURRENT(RUNDATE=&SYSDATE);
  %LET MONTHO=%SUBSTR(&RUNDATE,3,3);
  %LET YEARO=%SUBSTR(&RUNDATE,6,2);
  %LET MONO=1;
  %LET MLIST=JAN FEB MAR APR MAY JUN JUL AUG
    SEP OCT NOV DEC;
  %DO %WHILE(%SCAN(&MLIST,&MONO) ^= &MONTHO);
    %LET MONO=%EVAL(&MONO+1);
  %END;
%MEND CURRENT;
```

Figure 2. Retrieving Dynamic Information

Figure 2 illustrates a macro code using the list approach to retrieve month and year information dynamically. After compiling this code, we can get three global macro variables *MONO*, *YEARO*, and *MONO*. *MONO* is the base time point for the updating process. This time point can be used to

find the index and refer to the needed information later on.

The basic user interface can be set up during the design and development of the utility macros. Basically keyword parameters are the major tool to do this. In the routine maintenance run we try to avoid the user intervention as few as possible so that we can get more accurate and efficient production run. However, in some cases, the user has to be involved in the production process, for example, changing the default setting. It is more convenient to set default values for macro variables using keyword parameters. In this case we set *RUNDATE* as a default value *&SYSDATE*. The use of positional parameters should be avoided in end-user interfaces.

Step2. Extract the Needed Information

In order to extract the needed update information, we have to adjust the base time point to find the information index. The current time point is represented by the macro variable *MON0*. If the information index is *t* units, for example, *t* months or *t* years, away from the current time point, the adjustment that should be made to the *MON0* is *t*. The value of *t* depends on the information system. In most cases, *t* equals 1 since it is very common for a production system to process data from the previous month.

```
%MACRO CURRENT(RUNDATE=&SYSDATE);
...
%LET MINDEX=%EVAL(&MON0-t);
%LET MMINDEX=%SCAN(&MLIST,&MINDEX);
...
%
%MEND CURRENT;
```

Figure 3. Adjust the Current Time Point

Based on the values of *MON0* and *MINDEX*, we can obtain all the update information associated with this index.

Step 3. Organize the Requested Information

In fact the updating of the information system is to change the input/output file names, time periods, representation of the most current change, and other information. All such information can be created or organized from the update information obtained from the information stack. This can be done by using the macro concatenation techniques. For more details on this subject, please see the case study in the next section.

Validation Checking

After the coding is done, it is needed to start validating and testing the utility macros. For the monthly production system, it is easy to validate the possible results since there are only 12 possible inputs. The purpose of this kind of testing is to identify any errors and inconsistencies that exist in a system. Since the keyword parameter is set up for a user interface, randomly picking up a date value in each month for the keyword parameter *RUNDATE* can give an output for that month.

V. A Case Study

A production information system is built upon the existing historical data, i.e. the master files, while adding new data, i.e. the transaction files, which would be maintained on a monthly level. In order to update the system, we have to change the file names, variable names, or values stored in the names if necessary. All these updates can be done using the macro variables. One of the production programs has the following monthly updates (see Figure 4).

```
//PROGRAM1 JOB accounting-information
// EXEC SAS
//IN1 DD DSN=ABCCOMP.DEPT1.PROJECT1.FILE0897,
// DISP=SHR
//IN2 DD DSN=ABCCOMP.DEPT1.PROJECT1.DATSEP97,
// DISP=SHR
//OUT1 DD DSN=ABCCOMP.DEPT1.PROJECT1.FILE0997,
// UNIT=SYSDA,DISP=(,CATLG,DELETE),
// VOL=SER=XXXXX,
// SPACE=(TRK,(1000,100),,RLSE)
//OUT2 DD DSN=ABCCOMP.DEPT1.PUB.AUG97,
// UNIT=SYSDA,DISP=(,CATLG,DELETE),
// VOL=SER=XXXXX,
```

```
// SPACE=(TRK,(1000,100,),RLSE)
//SYSIN DD *

%LET BEGDATE=01SEP96; /* STARTING DATE */
%LET ENDDATE=31AUG97; /* ENDING DATE */
%LET MYTITLE=MONTHLY REPORT FOR THE MONTH
                OF AUGUST, 1997;
%LET INFILE1=FILE0897; /* Input File name */
%LET INFILE2=DATSEP97; /* Input File Name */
%LET OUTFILE1=FILE0997; /* Output File Name */
%LET OUTFILE2=AUG97; /* Output File Name */
....
```

Figure 4. An Example for the Case Study

Data sets stored in these libraries are named according to the date the data reflect. For example, DATSEP97 is for the monthly transaction data in September, 1997, FILE0897 for the master file in August, 1997, etc. The system runs monthly using data from the previous months. For example, the system runs in September, 1997 and creates new master files for August, 1997 using August master files and September transaction files.

This is just one program in the production system. All other programs have the similar monthly updates. If there are 20 programs in the production system and each program has 10 updates, including file names, time periods, and other updates, we have a total of 200 updates each month. In order to update this complex system accurately and efficiently, we can develop utility macros to update this system dynamically and internally. The remainder of this section will review the update requirements and build the utility macros associated with them.

Two utility macros will be created for this example. One is to update all variable names, values, and time periods. The other one is to update the file names. It is assumed that the base time point is SEP97. In order to get all information, we choose the index as AUG97.

Update Time Periods/Values

Figure 5 shows a utility macro which updates the time periods and texts.

```
%MACRO PERIOD(RUNDATE=&SYSDATE);
%LET MONTH0=%SUBSTR(&RUNDATE,3,8);
%LET YEAR0=%SUBSTR(&RUNDATE,6,2);
%LET MONO=1;
%LET MLIST=JAN FEB MAR APR MAY JUN JUL AUG
SEP OCT NOV DEC;
%DO %WHILE(%SCAN(&MLIST,&MONO) ^= &MONTH0);
%LET MONO=%EVAL(&MONO+1);
%END;
%LET INDEX=%EVAL(&MONO-1);
%LET MONTH1=%SCAN(&MLIST,INDEX);
%LET YEAR1=%EVAL(&YEAR0-1);

/* CREATE THE ENDING DATE;
%IF &INDEX=1 OR &INDEX=3 OR &INDEX=5
OR &INDEX=7 OR &INDEX=8 OR &INDEX=10
%THEN %DO;
%LET ENDDATE=31&MONTH1&YEAR0;
%ELSE %IF &INDEX=4 OR &INDEX=6 OR
&INDEX=9 OR &INDEX=11 %THEN %DO;
%LET ENDDATE=30&MONTH1&YEAR0;
%END;
%ELSE %IF &INDEX=2 %THEN %DO;
%LEAPYEAR(YEAR=&YEAR0)
%IF &LEAP=0 %THEN
%LET ENDDATE=28&MONTH1&YEAR0;
%ELSE ENDDATE=29&MONTH1&YEAR0;
%END;
%ELSE %IF &INDEX=0 %THEN %DO;
%LET ENDDATE=31DEC&YEAR1;
%END;

/* CREATE THE BEGINNING DATE;
%IF &INDEX=0 %THEN %DO;
%LET BEGDATE=01JAN&YEAR1;
%END;
%ELSE %DO;
%LET MONTHM=%SCAN(&MLIST,&MONO);
%LET BEGDATE=01&MONTHM&YEAR1;
%END;

/* CREATE TITLE FOR THE RPEORT;
%IF &INDEX=0 %THEN %DO;
%LET MYTITLE=QUARTERLY REPORT FOR THE 4TH
                QUARTER OF &YEAR1;
%END;
%ELSE %IF &INDEX=2 OR &INDEX=4 OR
&INDEX=5 OR &INDEX=7 OR &INDEX=8 OR
&INDEX=10 OR &INDEX=11 %THEN %DO;
%LET MYTITLE=MONTHLY REPORT FOR THE
                MONTH OF &MONTH1 &YEAR0;
%END;
%ELSE %IF &INDEX=3 %THEN %DO;
%LET MYTITLE=QUARTERLY REPORT FOR THE
                1ST QUARTER OF &YEAR0;
%END;
%ELSE %IF &INDEX=6 %THEN %DO;
%LET MYTITLE=QUARTERLY REPORT FOR THE
                2ND QUARTER OF &YEAR0;
%END;
%ELSE %IF &INDEX=9 %THEN %DO;
```

```
%LET MYTITLE=QUARTERLY REPORT FOR THE
3RD QUARTER OF &YEAR0;
%END;
%MEND PERIOD;
```

Figure 5. Update Time Periods/Values/Texts

The macro call %LEAPYEAR (shown in Figure 6) is used to identify whether a year is a leap year or not.

```
%MACRO LEAPYEAR(YEAR=);
DATA ONE;
YEAR=&YEAR*1;
IF MOD(YEAR,4)=0 THEN LEAPYEAR=1;
ELSE LEAPYEAR=0;
CALL SYMPUT ('LEAP', LEAPYEAR);
RUN;
%MEND LEAPYEAR;
```

Figure 6. A Macro to Define a Leap Year

Update Input/Output File Names

On MVS, we may allocate a SAS data library internally or externally. To allocate a SAS data library externally, use either a JCL statement or TSO command. To allocate it internally, use the LIBNAME statement. In order to update the input and output file names dynamically and internally, we need to allocate a SAS library internally.⁴

Figure 7 illustrates a utility macro to update file name. Knoop (1993) gave a good discussion on this subject. The present paper will extend his idea by using SYSDATE to create file names for the data sets on disks/tapes.

```
%MACRO UPDATE(RUNDATE=&SYSDATE);
%LET MAINDIR=ABCCOMP.FINANCE.DEPT1.PROJECT1;
%LET MAINDIR1=ABCCOMP.DEPT1.PUB;
%LET MONTH0=%SUBSTR(&RUNDATE,3,3);
%LET YEAR0=%SUBSTR(&RUNDATE,6,2);
%LET YEAR1=%EVAL(&YEAR0-1);
%LET MON0=1;
%LET MLIST=JAN FEB MAR APR MAY JUN JUL AUG
```

⁴ When data sets are stored in the SAS libraries on disks (or tapes), there may exist two possibilities. We may have only one data set stored in each of these disk libraries. Or we may have more than one data set stored in each of these disk libraries. We only consider the first case in this example.

```
SEP OCT NOV DEC;
%DO %WHILE(%SCAN(&MLIST,&MON0) ^= &MONTH0);
%LET MON0=%EVAL(&MON0+1);
%END;

%* INFILE1;
%IF &INDEX=0 %THEN %DO;
%LET IN1NAME=FILE12&YEAR1;
%END;
%ELSE %IF &INDEX=11 OR &INDEX=10 %THEN %DO;
%LET IN1NAME=FILE&INDEX&YEAR0;
%END;
%ELSE %DO;
%LET IN1NAME=FILE0&INDEX&YEAR0;
%END;
%LET IN1FILE=&MAINDIR..&IN1NAME;

%* INFILE2;
%LET FNAME=%SUBSTR(&RUNDATE,3,5);
%LET IN2NAME=DAT&FNAME;
%LET IN2FILE=&MAINDIR..&IN2NAME;

%* OUTFILE1;
%IF &MON0=10 OR &MON0=11 OR &MON0=12 %THEN
%DO;
%LET OUT1NAME=FILE&MON0&YEAR0;
%END;
%ELSE %DO;
%LET OUT1NAME=FILE0&MON0&YEAR0;
%END;

%* OUTFILE2;
%IF &INDEX=0 %THEN %DO;
%LET OUT2NAME=DEC&YEAR1;
%END;
%ELSE %DO;
%LET MON2=%SCAN(&MLIST,&INDEX);
%LET OUT2NAME=&MON2&YEAR0;
%END;
%LET OUT2FILE=&MAINDIR1..&OUT2NAME;

LIBNAME &IN1NAME "&IN1FILE" DISP=SHR;
LIBNAME &IN2NAME "&IN2FILE" DISP=SHR;
LIBNAME &OUT1NAME "&OUT1FILE" UNIT=SYSDA
DISP=NEW VOL=XXXXX
SPACE=(TRK,(1000,100,),RLSE);
LIBNAME &OUT2NAME "&OUT2FILE" UNIT=SYSDA
DISP=NEW VOL=XXXXX
SPACE=(TRK,(1000,100,),RLSE);

%MEND UPDATE;
```

Figure 7. Update Input/Output File Names

In order to invoke these two utility macros in the main program, we put them in a central library using the stored compiled macro facility (see Figure 8).

```

//PROGRAM1 JOB accounting-information
// EXEC SAS
//MACROLIB DD DSN=$USER.STORED,DISP=SHR
//SYSIN DD *

OPTIONS MSTORED SASMSTORE=MACROLIB;

%PERIOD(RUNDATE=&SYSDATE)
%UPDATE(RUNDATE=&SYSDATE)
other SAS statements

```

Figure 8. An Example for the Case Study

VI. Conclusion

Dynamic and internal updates provide an operator of the production system with significant efficiency, consistency, and accuracy while minimizing the update requirements associated with program maintenance. The techniques discussed here will give us a reduced code maintenance; as a result, we will find significant time savings from reduced program maintenance. These techniques shown in this paper can reduce or eliminate the need for date-related program maintenance. So this can help us achieve greater programming and maintaining efficiency, accuracy, and consistency.

It should be noted that effective and flexible macros do take considerably more time to code, document, and test. Given the benefits of these internal and dynamic updates, though, the extra time spent developing a utility macro is well worth the effort. This kind of efforts can save immense programming time and typing effort by simplifying tasks. As a result, it can drastically reduce the amount of maintenance needed by a macro-based information system.

Trademarks

SAS is a registered trademark of the SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Reference List

Huffman, Cam; Arneson, Carl P.; Horton, John; and King, John H., (1993) "Using the SAS System in the Pharmaceutical Industry: Project Setup and Standardization for the Analysis of Clinical Trial Data," in *Proceedings of the Eighteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

Knoop, Jim C. (1993) "Putting the Pieces Together: Applying Advanced SAS Macro Techniques to Everyday Problems," in *Proceedings of the Eighteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

Langston, Richard D. (1997) "The Year 2000: Preparing for the Inevitable," in *Proceedings of the Twentieth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

Levine, Howard, (1989) "Building Macro-Based Systems," in *Proceedings of the Fourteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

MacHose, William R., (1989) "Make a date with the SAS System," in *Proceedings of the Fourteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

Michel, Denis, (1994) "Maintenance Free Code Using the Contents Procedure," in *Proceedings of the Seventh Annual NorthEast SAS Users Group Conference*, Cary, NC: SAS Institute Inc.

Newhouse, Russell, (1997) "Validation and SAS Programming: Benefits of Using the System Life Cycle Method," in *Proceedings of the Twenty-Second Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

advanced tutorials

SAS Institute Inc., (1997). "SAS Macro Language: Reference: First Edition," (Cary, NC: SAS Institute Inc.).

SAS Institute Inc., (1990) "SAS Guide to Macro Processing, Version 6, Second Edition," (Cary, NC: SAS Institute Inc.).

Any questions or comments regarding the paper may be directed to the author:

Jeff F. Sun
Health Services Information And Analysis
Blue Cross Blue Shield of North Carolina
ET 600, 3rd Floor
Durham, NC 27705
Phone: (919)-765-7854
Fax: (919)-765-7840
E-mail: jeff_sun@scp11.bcbsnc.com