The Management Information Systems Department (MIS) at SAS Institute supports production systems running under both CMS and TSO. When new versions of SAS® products are developed, we use them in our production systems as soon as possible. This gives us the benefits of enhancements to existing products and the availability of new products for our systems. It also allows MIS to test products for the development groups at the Institute. MIS has converted its computer-based systems under releases of SAS software prior to Version 5 into Version 5.

What we hope to accomplish in this paper is to present a discussion on how the MIS Department converted its SAS software computer systems from 82.3 to Version 5, we will at the same time be presenting some of the uses of SAS software at SAS Institute.

It is not our purpose to present a comprehensive discussion of the SAS software uses at SAS Institute; instead we want to introduce the software while concentrating on the conversion process, the guidelines developed and followed for the conversion, and some tools used to simplify the conversion.

The SALES system is one of our menu driven systems written using the SAS software version 82.3. It is this interactive system we will use to demonstrate the steps taken in the conversion process. The SALES system is one of our larger systems on CMS and was designed for the Software Sales Department at SAS Institute. The system is invoked by using the SALES EXEC. The SALES EXEC calls in a SAS program that initiates a full-screen interactive environment providing a means of retrieving, displaying, inserting, deleting, and updating data. In addition to data manipulation, the Software Sales representatives rely entirely on this system to assist them in batch processing, report writing, and other numerous business and personal applications.

The design approach we had taken utilizes a single SAS program called SALES. When the driver EXEC SALES is invoked, SALES SAS was pulled in with the INCLUDE feature of the SAS System. The SALES program was responsible for defining the macro _MSG (the code that generated the menu screen), invoking the SETPFEXEC EXEC (whose only purpose was to define the PF keys, setting them to a macro name), and defining the macros that had been assigned to the PF keys.

With the exception of the _MSG macro, all of our macros consisted of a %INCLUDE statement. Modularizing the code using the %INCLUDE statements was necessary for ease of maintenance. The _MSG macro was unique for each system, whereas the same source code was used across several systems when user tasks overlapped. For example, all of our users need to access the Installation Data Base. The SAS code used to browse this data base consists of one single program accessed by more than 24 systems. Using the same code from a single source in different systems was simple to do and greatly reduced maintenance time.

The primary objective of our conversion project was to enhance all systems and increase efficiency while keeping the conversion as simple and inexpensive for the MIS Department as possible. In addition, we wanted the changes to be subtle. We did not want the user interface and the flow and structure of the systems to change in any form. We wanted to keep everything as "familiar" to the user as possible.

We were able to enhance the system simply by informing the users of the many new features in Version 5 SAS/FSP® software and by implementing several of the features by PF key selection. A particular feature used is the INCLUDE statement for PROC FSLETTER. By sending files to the reader, "receiving" them on the A disk, and using the INCLUDE statement while in PROC FSLETTER, staff members are able to pass letters and data among one another.

We achieved our objective of increased efficiency by implementing the macro autocall facility. With the use of this facility we no longer had to define all our macros in the primary program. The macro autocall facility allows the programmer to store macro source statements in a maclib, associate the maclib with the SAS session, and invoke the macro as needed. When a macro is called and the SAS system option MACROSOURCE is in effect (this option can be specified at SAS invocation or in an OPTIONS statement), the system will search the appropriate maclib (either SASAUTOS MACLIB or any other maclib you specify in the filedef) for a member with the same name as the macro called. When the SAS system finds the member, it is loaded and executed.

Our last objective, keeping the system familiar to the user, was accomplished by leaving the code that generates the menu screen unchanged. That is, we did not change the _MSG macro. Having the same menu screen in Version 5 as 82.3 helped the users adjust to the new version by keeping the obvious familiar.
Being that we had approximately 150 users and
more than 24 systems running under eMS to
convert, we found, for organizational purposes,
that it would be best to convert one system at a
time. Starting with a small system, we
documented the steps necessary to convert it.
Becoming familiar with the conversion process
and ironing out a lot of the problems on the
smaller systems made for easier conversion of
the larger, more complex systems.

For CMS, we began the conversion project by
defining a "new" production minidisk. In
defining this new production minidisk we were
able to keep the 82.3 version intact by moving
the program code to the new minidisk before
converting it to Version 5. Having the 82.3
version untouched allowed us to be able to have
both systems (82.3 and Version 5) running at the
same time. We moved several users over to
Version 5 while keeping the majority of our
users on 82.3. Once we were confident that the
entire system was functioning correctly, we
moved the rest of the users over. To "move" the
user over all that was necessary was to detach
the user from the old production minidisk and
link him to the new one.

Three additional points should be emphasized
here. First, having the new production minidisk
was extremely helpful in keeping the conversion
organized and error free. Second, it made it
easy for the MIS staff to determine whether a
given user was running on Version 5 or not,
simply by checking to see to which production
minidisk they were linked (the "new" Version 5
or the "old" 82.3). Third, having the new
minidisk on which to design and test alleviated
any problems which can arise from changing code
that users are currently accessing.

As mentioned earlier, we achieved our objective
of increased performance of the systems by
implementing the autocall facility. With this
feature in effect we were able to reduce a lot
of unnecessary macro compiling because now only
those macros the user invokes are compiled.
Before our conversion, all macros were compiled,
even though a given user may need only one or
two macros during a particular session. Besides
being very efficient in macro compiling, the
autocall facility requires less overhead than
using %INCLUDE statements to bring in source
code containing macro definitions.

Other significant benefits of using the autocall
facility, other than improved performance,
include simplification of the system design
process, maintenance, and support resulting from
having all the program code reside in one
location, the maclib. Implementing modularity,
by structuring the systems using the autocall
facility, provided the developers a single
reference point for all system code. As a
result, complexity of the systems was reduced.

To implement the autocall facility, it was
necessary to move all our SAS source code from
the minidisk it resided on into a maclib.
Before we moved the source code as individual
macros into the maclib it was necessary to do
the following to be compatible under Version
5.08.

1. Remove OPT=1 or OPTION=1 from all PROC
FSEDIT and PROC FSBRWNSE statements.

2. Change DATA= option to LETTER= on all PROC
FSLETTER statements.

3. Enclose the text portion of TITLE
statements in single quotes.

4. Enclose the text portion of LABEL
statements in single quotes.

Although items 3 and 4 will not cause an error
when executed under Version 5 (with the TEXT82
option on), they may cause errors in subsequent
versions of the SAS System.

In addition, it was necessary to convert all
FSLLETTER data sets, FSEDIT screens, and FSCALC
data sets using PROC FSCON.

Not wanting to spend a great amount of time on
CMS locating SAS programs that needed to be
changed, we used a utility EXEC called FINDFILE.
Depending upon what parameters you pass, the
FINDFILE EXEC searches your minidisk for a
particular string in a particular file. There
is a similar utility under TSO called IPOOPDTE.
This utility is only available from IBM if your
site has the CBIP0 (Custom Build Installation
Productivity Offering).

In addition to the tasks of modifying the source
code and then moving it to the maclib, we also
had to make various changes to the SALES EXEC
and the SETPFKEY EXEC.

The changes made to the SALES EXEC include
adding the MAUTOSOURCE system option on the
statement at which we invoke SAS and changing
the %INCLUDE SALES statement to %SALES (a macro
stored in the SASAUTOS MACLIB).

By removing the %INCLUDE statements from the
%SALES macro and making those programs
individual macros in the production maclib, the
%SALES macro became responsible only for
defining the %MSG macro (the screen definition
called _MSG in 82.3) and executing the SETPFKEY
EXEC.

The SETPFKEY EXEC was changed to define each PF
key as a macro name; the same name as it existed
as a member in the maclib. All the macros
defined by the SETPFKEY EXEC are stored in the
production maclib.
When more than one person works on development of software for a single system, certain problems frequently arise, one of which occurs when several people want to update a given component of a system. They may all make private copies, make the changes, and then attempt to install the modified code. Unless steps are taken to avoid it, only those changes made by the last person will be installed while the changes made by the others will be lost. To avoid this problem the MIS department has developed guidelines to be followed when a programmer is updating or developing source code. For the purpose of this paper we will only discuss those guidelines we have developed for updating macros residing in a maclib.

To begin with, MIS edited the SAS EXEC on its system to reference three maclibs (SASTEST, SASTEMP, and SASAUTOS). We concatenate these three maclibs to accommodate our need for:

1. a test library for testing all changes before putting into production.
2. a temporary library consisting of daily changes only.
3. a production maclib that is edited by no one.

Following we will explain the purpose of each maclib and the guidelines used for updating them.

**SASTEST MACLIB**

SASTEST is used exclusively for testing purposes. It is an empty (dummy) maclib residing on the production minidisk. As stated earlier, we edited our SAS EXEC so that the SAS System would search this maclib first for any macro that is invoked and not defined in the program code. A programmer wanting to test a macro would copy the empty SASTEST maclib from the production disk to his A disk and copy the macro they want to test from their A disk into the maclib. (We have some EXECs written for this particular purpose that we will discuss later.) By executing SAS with the MAUTOSOURCE system option and calling the macro, the programmer can easily test the changes that have been made. At the same time, the user who is linked to the production disks and would not want to pick up test data, is instead referencing the empty SASTEST maclib (assuming they don’t have a SASTEST maclib on their A minidisk).

**SASTEMP MACLIB**

SASTEMP maclib is used exclusively for putting new code into production during the day. SASTEMP maclib is the second maclib searched when a macro is called. When a macro is invoked SAS will first look in the SASTEMP maclib and if there is no member with the same name as the macro called, the SAS System will go on and search the SASTEMP maclib.

The developer, after testing new code in the SASTEST maclib, will log onto the production userid (which is the owner of the minidisks the maclibs reside on) and put the changes into the SASTEMP maclib. At night MIS runs a job that updates the production maclib (SASAUTOS) with the changes from SASTEMP. The job ends by clearing out the SASTEMP maclib so the programmer has an empty MACLIB to work with the next day.

There are two major advantages to updating the SASTEMP maclib rather than updating the production maclib (SASAUTOS). First, the SASTEMP maclib is much smaller than the master maclib, making it easier and faster for the programmer to work with. Updating and replacing a large maclib can be time consuming. Second, it is more likely that the production maclib could be accidently deleted if there are frequent updates made to it by many different developers.

**SASAUTOS MACLIB**

SASAUTOS is the production maclib. All updated macros and new code will end up in this maclib. The maclib is large, and replacing it takes a great deal of time. Therefore, we work with the SASTEMP maclib (which is empty each morning) and update SASAUTOS from SASTEMP each evening with a production job.

Here we introduce some EXECs developed to assist in retrieving and adding members to the maclibs. The following EXECs are much faster than using SPF to edit the maclib.

**GETMEM EXEC** extracts a given member from a given maclib and puts it on your A disk with a FT of SAS. If the member does not exist in the maclib, CMS will return an error "open error code 08."

**ADDMEM EXEC** puts SAS code into a given maclib. If the maclib does not exist, it creates one.

**MADDMEM EXEC** is used while logged onto the production userid only. It will put SAS code into a given maclib and erase the code from your A disk. If the maclib does not exist, it does not create one. In addition, this EXEC renames the macro being replaced and puts it out on our production disk giving us a backup; a reference to the old code should we need it.

As an example, lets modify a macro named SALES. To retrieve a copy of the SALES SAS macro from the production maclib we will use the command:

```
GETMEM SASTEMP SALES
```
We check the SASTEMP maclib first because a programmer might have changed the SALES macro today. If CMS returns an "open error code 0" then we know the macro has not been updated today. Instead, we get the macro from the production maclib (SASAUTOS) with the command:

GETMEM SASAUTOS SALES

The GETMEM EXEC makes a copy of the macro SALES from SASAUTOS, putting it on the A disk and naming it SALES SAS (SALES for the name of the macro and it's always given a FT of SAS). We may now edit the SALES SAS program, making the necessary changes. When the changes are complete we would add it to the SASTEST maclib by using the command:

ADDMEM SASTEST SALES

ADDMEM EXEC makes a copy of the SALES SAS file on the A disk, putting the copy in the SASTEST maclib. It leaves the SALES SAS file on the A disk so that further changes can be made to the program code if necessary. To test the changed SALES macro in the SASTEST maclib, we execute SAS with the MAUTOSOURCE option and invoke the SALES macro. When finished testing, we would log onto the production userid (which is the owner of the minidisks that the maclibs reside on). Once on the production userid, we would link to the location of the SALES SAS program that was tested and use the CMS COPY command.

COPY SALES SAS * = = A

This will copy the SALES SAS program over to the production A minidisk. Using the CMS COPY command again

COPY SASTEMP MACLIB E = = A

we copy the SASTEMP maclib from the production disk to the A disk (we don't want to change the same maclib the users are accessing). We use the MADDEM EXEC to copy the SALES SAS program from the A disk into the SASTEMP maclib with the command:

MADDEM SASTEMP SALES

The MADDEM EXEC, before it updates the SASTEMP maclib, will store a copy of the SALES program on the production minidisk. We keep the "before changed" copy of the SALES macro in case we need to refer to it at a later date.

Last, we replace the newly updated SASTEMP maclib back to the production minidisk so the user can access the "latest" updates. Use the command:

REPLACE SASTEMP SALES

The REPLACE EXEC will copy the SASTEMP maclib over to the production disk after renaming the old one. All the user needs to do to is re-access the production disk to get the latest SALES macro.

The first sections of this session have dealt with conversion to the Version 5 SAS software with emphasis on our larger systems running under CMS. We also have many systems running under TSO that have been converted to Version 5.

Most of the essential changes for the conversion are the same on both systems, including the use of the macro.autocall facility. However, under TSO this facility is accessed through the use of a DDNAME of SASAUTOS to concatenate MIS macro libraries to the system library and having the MAUTOSOURCE system option in effect.

We were anxious to use SAS/AF® software; its design is particularly useful for production environments where "non-programmers" are using computers to perform more and more of their work load. Since the menu-driven displays we prefer to use were much more cumbersome under TSO than CMS, many of our systems running under TSO were still using CLISTS with continuous question-response style design. SAS/AF provided us with a good solution to this problem. We prefer menu-driven systems that display tasks provided for users in a picture of their function keys. This approach requires less expertise from users, making the system much easier to understand by providing a self-documenting element for them. Users know at a glance what applications are available for them. Menu systems provide users with a greater sense of what applications their systems provide, as well as the flexibility to perform tasks in varying sequences without going through a series of questions again and again. This gives them more control over processing.

SAS/AF software allows for additional customized documentation through programmer-written (MIS) HELP screens available to the user through the HELP function key or the HELP command. Here the MIS staff can provide on-line user documentation to supplement other more traditional styles of user documentation. Through the use of color display, which can easily be changed using the software, color-coded screens can also be used, making the similar tasks of editing, browsing, and submitting batch jobs all designated through color as well as words on the menu display screen.

Based on data entered by the user, PROC DISPLAY can be set up to conditionally execute applications by making use of the automatic macro variables, the conditional statements, and data checking options provided in the software. This feature makes a single program function adequately for various users doing multiple but similar tasks, again giving users greater flexibility and control over their individual needs. It also allows multiple functions to be consolidated in one program.
In order to keep up with the dynamic systems supported by the MIS staff, a set of standards was established and is being established on a continuing basis. These standards are followed when designing and implementing a new system, when converting a system to run under Version 5, and especially when implementing SAS/AF. These guidelines make our code consistent, predictable, and establish the "easy way" of programming as the required way. We have previously explained some of these guidelines as they apply to CMS, the use of our maclibs, creation of a Version 5 disk for each system, and the specialized EXECs for updating code.

Other rules apply to naming conventions. Naturally, guidelines instituted by our data center must be included in the MIS standards. The MIS naming conventions distinguish between batch programs, macro source code, CLISTs, SAS data libraries, SAS/AF utility data sets, and macro libraries. These unique names are important for ongoing support. In addition to controlling the naming of files, the names of program screens within PROC BUILD are important. Since we use the autocall facility, our macros are members of a PDS or maclib and their names must correspond to the macro name. We also use that same name as the program screen name in the SAS/AF product. When naming macros under CMS, remember to use a maximum of seven characters. Macros that perform the same tasks on systems use the same name and the same macro, such as the call to PROC DISPLAY and PROC BUILD, as well as applications that are universal. An example of one of these universally used macros is the program screen for browsing the Institute's phone directory. This way a single macro is used for many systems which in turn cuts out the overhead costs.

For a given system, no matter how many tasks might be needed, we use a single entry point into the SAS code, an EXEC or CLIST. Once in SAS, jobs are performed with function keys. When there are more tasks than keys, we add multiple screens that are accessed by one of the keys from the primary menu. All of the code for these tasks are SAS macros that are stored using the autocall facility. Having one entry point into SAS is important in reducing processing time and overhead costs.

On TSO we keep two versions of the CLISTs to enter the same SAS/AF production system. One version is used by the user and goes directly into a SAS/AF display procedure that brings up the system menu screen. The other is used by the MIS support staff when making changes and enhancements to the system. The MIS version uses options to allow notes and messages to be displayed, to display the menu screen, and to remain in the SAS System when exiting from the SAS/AF software. This allows re-entry into the SAS/AF screen or another program without re-invoking SAS.

In all of our systems, we have a number of PROC DISPLAY panels available that do not appear on the menu screen seen by the user. These programs are accessed by special numbers or letters entered on the command line instead of using a function key call. These commands are part of our universal system design. They include a "B" to execute PROC BUILD and an "M" to bring up a SAS/AF program screen which when filled in with a macro name reloads the macro into the autocall facility. Having these programs available for use on command makes system maintenance faster, which is an important element to successful system support.

While developing the guidelines for the department and experimenting with Version 5 and SAS/AF, we developed a skeleton menu system that includes the previously mentioned commands, two versions of a screen display, and the universal applications used in all our systems. When creating a new SAS/AF application, the programmer has a great head start. First he or she creates the data sets to be used named according to our convention, then copies the skeleton system into the appropriate data set. From there the programmer can enter the MIS version CLIST, use the "B" command and enter PROC BUILD with the name embedded in place. Adding the additional code for the new system is relatively straightforward, again saving time and keeping coding techniques consistent for future support.

Since we use the autocall facility so extensively within our SAS/AF panels, our code at this point is succinct. It usually consists of a single display, two calls to macros that are stored in a PDS. The first call brings in the code for the function needed by the user, the second recalls the display procedure. This brevity of code makes the naming convention for the source code an important rule to follow. Access into that PDS for editing can be made while still in the SAS session, and the reloading of the altered macro can be achieved by using the "M" command coded in all our SAS/AF systems.
When converting systems to Version 5, it was essential for the MIS department to keep the users informed of the conversion that was taking place. We sent them memos documenting the changes they would find in SAS/FSP software. We informed them of the enhancements to Version 5 over the prior release of SAS software they had been using. In addition, for departments with a large number of staff using systems supported by MIS, it was helpful to have one of the more experienced users designated as a trainer or leader for others in that department. As well as being the first person to use a system that had been converted, they served as a reference source to the rest of their department. This further reduced time spent by MIS staff to train users and provided another good means of communication between MIS and our users. By the completion of the conversion, the users were more than enthusiastic to be moved over and running under Version 5.

The MIS management had hopes of converting all SAS software menu systems to SAS/AF software. Unfortunately, when considering time constraints and resources necessary to convert all our menu systems, we found it wise to convert all current systems to Version 5 first. Then, as time permits we gradually convert them to SAS/AF. However, all new menu systems are being developed using SAS/AF software.

This paper has shown how converting your SAS software to Version 5 can be a simple and painless experience. With some thought-out design techniques and design tools, the conversion process will not require a great amount of resources or time. In addition, converting to Version 5 has considerable advantages for the purposes of organizing code, increasing performance and reduction of support time.

We hope that you will find some of these ideas helpful and adopt them for use in production systems at your site.
FINDFILE EXEC: Searches minidisk for a text string in files specified.

```plaintext
PARSE UPPER ARG FN FT FM PARM1;
IF FN="" FT="" THEN DO;
   SAY 'SYNTAX: FN FT FM STRING1 ';
   EXIT;
END;
/* EXPAND FN FT IN CASE THEY CONTAIN WILDCARDS */
'LISTFILE' FN FT FM 'E'
/* READ FIRST LIST IN FILE */
DO FOREVER;
   CALL READ 'CMS EXEC A';
   IF RC<>0 THEN LEAVE;
   SEARCHID=STRIP(RESULT,'S');
   /* GET RID OF &1 ANO &2 */
   PARSE VAR SEARCHID .. SEARCHID;
   SAY 'SEARCHING:' SEARCHID;
   /* SEE IF STRING IN FILE */
   DO FOREVER:
      CALL READ SEARCHID;
      IF RC<>0 THEN DO;
         CALL CLOSE SEARCHID;
         LEAVE;
      END;
      IF 0<>POS(PARM1,RESULT) THEN DO;
         END;  
      END;
      PUSH '&I' PARM1;
      XEDIT SEARCHID;
      CALL CLOSE SEARCHID;
      LEAVE;
   END;
   CALL CLOSE 'E';

REPLACE EXEC: Copies a file to the production disk after renaming any existing file of the same name.

```
MADDEM EXEC: Moves SAS code into a maclib after renaming any existing macro of the same name. Renamed macro is moved to the production disk.

```sas
/* THIS EXEC WILL PUT CODE OF FILETYPE SAS INTO A GIVEN MACLIB. */
/*/ SYNTAX: ADD FN MAC */
/*/ WHERE FN IS THE FILENAME OF THE PROGRAM */
/*/ FILETYPE IS SAS */
/*/ MAC IS THE NAME OF THE MACLIB */
/*---------------------------------------------------------*/
TRACE OFF

PARSE UPPER ARG FN MAC;
/* CHECK SYNTAX */
IF FN="" | MAC="" THEN DO;
  SAY 'SYNTAX: FN MAC';
  EXIT;
END;
/* CHECK MACLIB */
IF MAC="SASTEMP" THEN DO;
  SAY 'THIS EXEC SHOULD ONLY BE USED ON SASTEMP';
  EXIT;
END;
/* CHECK IF FILE EXISTS */
SET CMSTYPE HT
STATE FN SAS A
RCC:RC
SET CMSTYPE RT
IF RCC=O THEN DO;
  SAY 'WE CANNOT FIND THE FILE' FN SAS ' ON YOUR A DISK';
  EXIT;
END;
/* CHECK IF MACLIB EXISTS */
SET CMSTYPE HT
'STATE' MAC' MACLIB A'
RCC:RC
SET CMSTYPE RT
IF RCC=O THEN DO;
  SAY 'WE CANNOT FIND THE MACLIB ON THE A DISK' MAC;
  EXIT;
END;
/* CHECK IF FILE EXISTS ON THE A DISK WITH FILETYPE OF COPY */
SET CMSTYPE HT
STATE FN COPY A
RCC:RC
SET CMSTYPE RT
IF RCC=O THEN DO;
  SAY 'WE NEED TO RENAME YOUR SAS FILE TO HAVE A FILETYPE OF COPY';
  SAY 'SO THAT WE CAN ADD THE FILE TO THE MACLIB. YOU ALREADY ';
  SAY 'HAVE THIS FILE ON YOUR A DISK AND YOU NEED TO EITHER DELETE';
  SAY 'OR RENAME THIS FILE.';
  EXIT;
END;
/* COPY CURRENT SAS PROGRAM IN MACLIB AS BACKUP */
ACC 130 E
'FILEDEF INMOVE DISK ' MAC ' MACLIB * (MEMBER' FN
FILEDEF OUTMOVE DISK FN ZZZZZZ E
MOVEFILE
/* BACKUP */
ERASE FN O000SAS E
RENAME FN O000SAS E FN O000SAS E
RENAME FN O00SAS E FN O000SAS E
RENAME FN O0SAS E FN O0SAS E
/* RENAME CURRENT VERSION FROM MACLIB */
RENAME FN ZZZZZZ E FN O0SAS E
ERASE FN SAS E
COPYFILE FN SAS A FN SAS E
/* RENAME THE FILETYPE OF THE NEW SAS CODE TO COPY AND ADD TO MACLIB */
RENAME FN SAS A FN COPY A
MACLIB REP MAC FN
IF RC=O THEN DO;
  SAY 'THE SAS SOURCE CODE' FN ' WAS NOT IN SASTEMP MACLIB.';
  SAY 'THE SAS SOURCE CODE WAS ADDED TO THE SASTEMP MACLIB.';
END;
/* DISCARD FROM A DISK */
ERASE FN COPY A
FILEDEF INMOVE CLEAR
FILEDEF OUTMOVE CLEAR
'ACC 130 E/E'
```

460
GETMEM EXEC: Extracts a member from a maclib and puts a copy on your A disk with a filetype of SAS.

```sas
/* EXTRACT MEMBER FROM ANY MACLIB */
/* AND COPY TO YOUR A MINI DISK */
/* SYNTAX: GETMEM MEMBER MAC */
/* WHERE MEMBER IS THE MEMBER OF THE */
/* MAC (MACLIB) */
PARSE UPPER ARG MEMBER MAC
IF MEMBER=" | MAC=" THEN DO;
  SAY 'SYNTAX: GETMEM MEMBER MAC';
  EXIT;
END;
'STATE' MAC ' MACLIB *'
IF RC=4 THEN DO;
  SAY 'THE MACLIB ' MAC ' IS NOT FOUND';
  EXIT;
'FILEDEF INMOVE DISK MAC MACLIB ' MEMBER MEMBER' FILEDEF QUTMOVE DISK MAC SAS A' 'MOVEFILE FILEDEF INMOVE CLEAR' 'FILEDEF QUTMOVE CLEAR'

ADDMEM EXEC: Puts SAS code in a maclib. If the maclib does not exist, it creates one.

```sas
/* THIS EXEC WILL PUT CODE OF FILETYPE SAS INTO A GIVEN MACLIB */
/* */
/* SYNTAX: ADD FN MAC */
/* WHERE FN IS THE FILENAME OF THE PROGRAM */
/* FILETYPE IS SAS */
/* MAC IS THE NAME OF THE MACLIB */
```
AUTOS SAS: Nightly production job to update the SASAUTOS MACLIB with changed or new macros in the SASTEMP MACLIB.

/ * 1 will update the SASAUTOS MACLIB with changed or new macros */
/ * which reside in the SASTEMP MACLIB. This program will run nightly. */
/ * erase backups from the previous night */
%cms erase tempbk maclib e;
%cms erase autobk maclib e;
%cms maclib map sastemp;  /* creates a mapping of the maclib in */
/ * SASTEMP MAP A */
%cms find mac disk sastemp mac a;
data null; infile inmacro end=eof;
length mac $8.;
input mac;
%macro backup;
%copyfile sastemp maclib e tempbk maclib e: run;
%if &sysrc~=D %then %do;
%put ==========problem creating backup of sastemp ==========
%end;
%copyfile sasautos maclib e autobk maclib e: run;
%if &sysrc~=O %then %do;
%put ==========problem creating backup of sasautos============;
%end;
%mend backup;

%macro do i t;
%if &max>O %then %do ;=1 %to &max:
%cms exec getmem &mac&i sastemp; run;
%cms exec addmem &mac&i sasautos: run;
%mend do;
%if &max=&i %then %do;
%cms maclib del sastemp &mac&i; run;
%mend do;
%else %put We did not add any new members in that SASTEMP was empty;
%mend doit;

Two versions of a CLIST to enter the same SAS/AF production system, one used by MIS staff and one used by our users.

PROC O
ALLOC F(SASUSER) NEW SP(5 0) TRACK REUSE
ALLOC FI(AUTOS) DA('SASADM3.EQUIP.MACROS') SHR REUSE
ALLOC FI(AUTOS2) DA('SASMIS.GEN.MACROS') SHR REUSE
CONCAT (AUTOS AUTOS2)
ALLOC FI(IN) DA('SASADM3.EQUIP.BUILD') OLD REUSE
SET INT=&STR(%DSTAND; RUN;)
SET OPT=&STR( NONEWS S=72 NOOMS MPRINT SYMBOLGEN)
SET OPT2=&STR(SASAUTOS=AUTOS MACRO LOG=X)
SAS OPTIONS('&OPT2 &OPT INITSTMT="∫ " ,)
FREE ALL

PROC O
ALLOC F(SASUSER) NEW SP(5 0) TRACK REUSE
ALLOC FI(AUTOS) DA('SASADM3.EQUIP.MACROS') SHR REUSE
ALLOC FI(AUTOS2) DA('SASMIS.GEN.MACROS') SHR REUSE
CONCAT (AUTOS AUTOS2)
ALLOC FI(IN) DA('SASADM3.EQUIP.BUILD') SHR REUSE
SET INT=&STR(%DSTAND; RUN;)
SET OPT2=&STR(SASAUTOS=AUTOS MACRO LOG=X)
SAS OPTIONS('&OPT2 &OPT INITSTMT="∫ " ,)
FREE ALL
Catalog displayed using PROC BUILD for the skeleton SAS/AF system.

Directory for Catalog: IN.STANDARD

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASTER1</td>
<td>MENU</td>
<td>PF key picture menu</td>
<td>28JAN86</td>
</tr>
<tr>
<td>MASTER2</td>
<td>MENU</td>
<td>PF key list menu</td>
<td>15FEB85</td>
</tr>
<tr>
<td>BUILD</td>
<td>PROGRAM</td>
<td>for use by MIS only</td>
<td>15NOV84</td>
</tr>
<tr>
<td>INFO</td>
<td>PROGRAM</td>
<td>browse Institute information</td>
<td>15NOV84</td>
</tr>
<tr>
<td>ENDSAS</td>
<td>PROGRAM</td>
<td>exit DISPLAY and SAS</td>
<td>03NOV84</td>
</tr>
<tr>
<td>INSTALL</td>
<td>PROGRAM</td>
<td>browse installation data base (IDB)</td>
<td>15DEC85</td>
</tr>
<tr>
<td>PHONE</td>
<td>PROGRAM</td>
<td>browse phone list</td>
<td>15DEC85</td>
</tr>
<tr>
<td>QUERY</td>
<td>PROGRAM</td>
<td>check status of jobs</td>
<td>03NOV84</td>
</tr>
<tr>
<td>SASCOM</td>
<td>PROGRAM</td>
<td>browse SASCOM mailing list</td>
<td>03NOV84</td>
</tr>
<tr>
<td>SASLOG</td>
<td>PROGRAM</td>
<td>enter SAS log</td>
<td>03NOV84</td>
</tr>
<tr>
<td>SETMAC</td>
<td>PROGRAM</td>
<td>reset a macro that has been changed</td>
<td>20AUG84</td>
</tr>
</tbody>
</table>

PF key picture menu for the skeleton SAS/AF system (IN.STANDARD.MASTER1.MENU).

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info</td>
<td>Phone list</td>
<td>Jobs Display</td>
<td>Reset a macro</td>
<td>Browse IDB</td>
<td>Browse Sascom</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

STANDARD INFORMATION SYSTEM

QUIT
Attribute screen for PF key picture menu (IN.STANDARD.MASTER1.MENU).

### Attributes for MENU & CST screens

<table>
<thead>
<tr>
<th>Option</th>
<th>Key</th>
<th>Name</th>
<th>Type</th>
<th>Libref</th>
<th>Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>INFO</td>
<td>PROGRAM</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>PHONE</td>
<td>PROGRAM</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>QUERY</td>
<td>PROGRAM</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>M</td>
<td>4</td>
<td>SETMAC</td>
<td>PROGRAM</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>INSTALL</td>
<td>PROGRAM</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>SASCOM</td>
<td>PROGRAM</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>BUILD</td>
<td>PROGRAM</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td>SASLOG</td>
<td>PROGRAM</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>ENDSAS</td>
<td>PROGRAM</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Program screen for Option M (Reset a macro) in the skeleton SAS/AF system (IN.STANDARD.SETMAC.PROGRAM).

Please enter the name of the SAS macro that has been changed: &macro

```
%inc #MACRO sasautos(&macro);
%dstand; run;
```