Paper 153-26

An Introduction to SAS Macros

Steven First, Systems Seminar Consultants, Madison, WI

Abstract

The SAS programming language has a rich "tool-box" of features that can offer a lot of power to the user. The use of macro variables and macro code is difficult to learn without some training. This workshop will introduce the user to a few of the very powerful techniques possible using macro variables and macro programming. The user will also see examples that are easy to understand

Some of the topics to be covered include:

1. What is a macro variable and what does it do?
2. Examples of some of the standard SAS macro variables
3. How the SAS macro processor handles macro variables
4. Passing information to other steps in a program using macro variables
5. What is a macro?
6. Creating and invoking macro code
7. How to "conditionally" run SAS steps based on the logic in your program
8. Changes and Enhancements to the macro language in Version 7 & 8

Introduction

This paper corresponds to the hands-on workshop presented at SUGI. The SAS macro facility can be somewhat daunting due to it's complexity and comprehensive nature. However considerable benefit can be gained by using some of the simple features of the facility along with an introductory understanding of macro structures and timing issues. This paper and the workshop will attempt to start that understanding..

SAS Macro Overview

The SAS Macro facility constructs source programs to be input to the SAS compiler just as pressing keys does when we write a program.

Some functions of the SAS macro processor are to pass symbolic values between SAS statements and steps, to establish default symbolic values, to conditionally execute SAS steps, and to hopefully invoke very long, complex code in a quick, short way.

It should be noted that the MACRO PROCESSOR is the SAS system module that processes macros and the MACRO LANGUAGE is how you communicate with the processor.

Traditional SAS Programming

Without macros, things that you cannot easily do are to substitute text in statements like TITLEs, to communicate across SAS steps, to establish default values, to conditionally execute SAS steps, and to hide complex code that can be invoked easily.

The macro facility will allow us to do the above and more.

Traditional Flow Without Macros

Without macros, SAS programs are DATA and PROC steps. The program is scanned one statement at a time looking for the beginning of step (step boundary). When the beginning of step is found, all statements in the step are compiled one at a time until end of step is dectected. When the end of step is found (the next step boundary), the previous step executes.

SAS Step Boundaries

The following keywords are step boundaries to the SAS system:

DATA	ENDSAS
PROC	LINES
CARDS	LINES4
CARDS4	PARMCARDS
DATALINES	QUIT
DATALINES4	RUN

RUN acts as an explicit step boundary in most PROCs. It is highly recommended that the RUN statement be coded, as there then no question to when compile finishes.

```
data saleexps;                  <--Step, start compile
 infile rawin;
 input name $1-10 division $12
       years 15-16 sales 19-25
       expense 27-34;
 run;                           <--Step end, exec
   previous

proc print data=saleexps;       <--Step start, start
   compile
 run;                           <--Step end, exec
   previous
proc means data=saleexps;
 var sales expense;
 run;                           <--Step end, exec
   previous
```

The SAS Macro Language

The Macro Language is a second SAS programming language for string manipulation. Characteristics of the language are:

- strings are sequences of characters
- all input to the macro language is a string
- usually strings are SAS code, but they don't need to be
- the macro processor manipulates strings and may send them back for scanning.

Macro Language Components

The macro language has several kinds of components.

Macro variables:

- are used to store and manipulate character strings
- follow SAS naming rules
- are NOT the same as DATA step variables
- are stored in memory in a macro symbol table.

Macro statements:

- begin with a % and a macro keyword and end with semicolon (;)
- assign values, substitute values, and change macro variables
- can branch or generate SAS statements conditionally.

Macro Processor Flow

Macro statements are given to the macro processor BEFORE the compiler where substitution may occur.

- Macro statements start with %.
- Macro variables are referred to with &.

A Macro Problem

Problem:
You would like to have the Day of Week and current date appear in a title, but SAS titles are text, not variables.

Solution:
Use some system macro variables.

```
PROC PRINT DATA=DEPTSALE;
 TITLE  "Department Sales as of &SYSDAY  &SYSDATE";
 TITLE2 "Deliver to Michael O'Malley";
RUN;
```

It should be noted that macro variables are NOT resolved within single quotes.

Automatic Macro Variables
A partial list of automatic macro variables and their usage are:

SYSBUFFR	text entered in response to %INPUT
SYSCMD	last non-SAS command entered
SYSDATE	current date in DATE6. or DATE7. format
SYSDAY	current day of the week
SYSDEVIC	current graphics device
SYSDSN	last dataset built (i.e.,WORK SOFTSALE)
SYSENV	SAS environment (FORE or BACK)
SYSERR	return code set by SAS procedures
SYSFILRC	whether last FILENAME executed correctly
SYSINDEX	number of macros started in job
SYSINFO	system information given by some PROCS
SYSJOBID	name of executing job or user
SYSLAST	last dataset built (i.e., WORK.SOFTSALE)
SYSLIBRC	return code from last LIBNAME statement
SYSLCKRC	whether most recent lock was successful
SYSMENV	macro execution environment
SYSMSG	message displayed with %DISPLAY
SYSPARM	value passed from SYSPARM in JCL
SYSPROD	indicates whether a SAS product is licensed
SYSPBUFF	all macro parameters passed
SYSRC	return code from macro processor
SYSSCP	operating system where SAS is running
SYSTIME	starting time of job
SYSVER	SAS version

Another example:

```
FOOTNOTE "THIS REPORT WAS RUN ON &SYSDAY, &SYSDATE";
```

Resolves to:

```
FOOTNOTE "THIS REPORT WAS RUN ON FRIDAY, 26MAR99";
```

Displaying Macro Variables
The %PUT statement can display individual or all macro variables to the log at compile time. Macro variables are prefixed with & and the special variable _ALL_ can display all known macro variables. Any other text is displayed unchanged.

Example:

```
DATA NEWPAY;
 INFILE DD1;
 INPUT EMP$ RATE;
RUN;
%PUT ***** &SYSDATE *****;
```

Partial SAS Log:

```
***** 26MAR99 *****
```

Displaying All Macro Variables

```
%PUT _ALL_;
```

Partial SAS Log:

```
GLOBAL MBILLYR 99
GLOBAL SSCDEV C
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 26MAR99
AUTOMATIC SYSDAY Tuesday
Partial SAS LOG Continued
AUTOMATIC SYSDSN          _NULL_
AUTOMATIC SYSENV FORE
AUTOMATIC SYSERR 0
AUTOMATIC SYSFILRC 0
AUTOMATIC SYSINDEX 1
AUTOMATIC SYSINFO 0
AUTOMATIC SYSJOBID 0000016959
AUTOMATIC SYSLAST _NULL_
AUTOMATIC SYSMSG
AUTOMATIC SYSPARM
AUTOMATIC SYSRC 0
AUTOMATIC SYSSCP WIN
AUTOMATIC SYSSCPL WIN_32S
AUTOMATIC SYSSITE 0011485002
AUTOMATIC SYSTIME 10:35
AUTOMATIC SYSVER 6.11
AUTOMATIC SYSVLONG 6.11.0040P030596
```

Another Macro Problem
You reference a SAS datasetname several times in a SAS job.

```
DATA PAYROLL;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=PAYROLL;
 TITLE "PRINT OF DATASET PAYROLL";
RUN;
```

Question: How can you change the name quickly in one place only and have the datasetname appear in a title?

Solution: Use a user defined macro variable.

Characteristics:
- You can define macro variables with %LET.
- You refer to the variables later with &variable.
- Macro will substitute value for all occurrences of &variable.

Syntax:

%LET variable=value;

Example:

```
%LET NAME=PAYROLL;
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated SAS Code

```
DATA PAYROLL;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
```

```
PROC PRINT DATA=PAYROLL;
 TITLE "PRINT OF DATASET PAYROLL";
RUN;
```

Note that because Macro variables are not resolved within single quotes, the double quote character is used and that leading and trailing spaces are discarded.

To assign a new value, code another %LET statement.

```
%LET NAME=NEWPAY;
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated SAS Code

```
DATA NEWPAY;
 INPUT EMP$ RATE;
DATALINES;
 TOM 10
 JIM 10
   ;
PROC PRINT DATA=NEWPAY;
TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

When assigning special characters (especially semicolon) the %STR function allows values with ; etc.

```
%LET NAME=NEWPAY;
%LET CHART=%STR(PROC CHART;VBAR EMP;RUN;);
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
&CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated SAS Code

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC CHART;VBAR EMP;RUN;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Nesting of Macro Variables is allowed by having macro variables contain references to other macro variables.

```
%LET NAME=NEWPAY;
%LET CHART=%STR(PROC CHART DATA=&NAME;VBAR EMP;RUN;);
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
&CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated SAS Code

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC CHART DATA=NEWPAY;VBAR EMP;RUN;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Other Macro Debugging Options

SAS gives several system options that control log output.

- SYMBOLGEN/NOSYMBOLGEN controls display of variable resolution
- MPRINT/NOMPRINT displays generated statements given to the compiler
- MLOGIC/NOMLOGIC displays tracing information during macro execution.

What is a Macro?
More than just macro variables and text, a SAS macro contains stored text that can be inserted anywhere in a SAS program and expanded including:

- constants such as literals, variables, names, and statements
- assignments to macro variables
- macro programming statements
- macro language functions
- invocations of other functions
- nested macro definitions
- LOGIC to conditionally generate SAS code.

Defining and Using Macros
%MACRO and %MEND define macros.

%macroname will invoke it later.

Example: Define a macro to run PROC CHART and later invoke

```
%MACRO CHART;
  PROC CHART DATA=&NAME;
    VBAR EMP;
RUN;
%MEND;
```

Invoking the Chart Macro

```
%LET NAME=NEWPAY;
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
RUN;
%CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated Code

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
RUN;
PROC CHART DATA=NEWPAY;
 VBAR EMP;
RUN;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Positional Macro Parameters

Macro parameters are defined in order after the macro name.

Keyword parameters can also be defined using = . Keywords can be specified in any order and they can also set default values.

```
%MACRO CHART(NAME,BARVAR);
  PROC CHART DATA=&NAME;
    VBAR &BARVAR;
  RUN;
%MEND;

%CHART(PAYROLL,EMP)
```

Resolves to:

```
PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
```

Nested Macros can be utilized when one macros calls another.

```
%MACRO CHART(NAME,BARVAR);
  PROC CHART DATA=&NAME;
    VBAR &BARVAR;
     RUN;
%MEND;

%MACRO PTCHART(NAME,BARVAR);
%CHART(PAYROLL,EMP)
  PROC PRINT DATA=&NAME;
    TITLE "PRINT OF DATASET &NAME";
  RUN;
%MEND;

%PTCHART(PAYROLL,EMP)
```

The Generated SAS Code

```
PROC CHART DATA=PAYROLL;
 VBAR EMP;
RUN;
PROC PRINT DATA=PAYROLL;
  TITLE  "PRINT OF DATASET PAYROLL";
RUN;
```

Conditional Macro Compilation

One of the most useful features of a macro, is that the programmer can use %IF and other statements to conditionally pass code to the compiler. While the DATA step has IF statements, %IF is the only practical way to run a PROC some of the time.

Example: Run PROC PRINT only if PRTCH=YES.

```
%MACRO PTCHT(PRTCH,NAME,BARVAR);
  %IF &PRTCH=YES %THEN PROC PRINT DATA=&NAME;
       ;
  PROC CHART DATA=&NAME;
   VBAR &BARVAR;
  RUN;
%MEND;

%PTCHT(YES,PAYROLL,EMP)
```

```
The Generated SAS Code
PROC PRINT DATA=PAYROLL ;

PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
```

The %DO Statement

%DO allows many statements to be conditionally compiled.

Example: Submit as before, but include titles.

```
%MACRO PTCHT(PRTCH,NAME,BARVAR);
  %IF &PRTCH=YES %THEN
      %DO;
          PROC PRINT DATA=&NAME;
          TITLE "PRINT OF DATASET &NAME";
          RUN;
      END;
   PROC CHART DATA=&NAME;
    VBAR &BARVAR;
   RUN;
%MEND;

%PTCHT(YES,PAYROLL,EMP)
```

The Generated SAS Code

```
PROC PRINT DATA=PAYROLL;
 TITLE  "PRINT OF DATASET PAYROLL";
RUN;
PROC CHART DATA=PAYROLL;
 VBAR EMP;
RUN;
Interactive Macro Invocation
%DO can also vary a value.

Example:  Run PROC PRINT &PRTNUM times.

%MACRO PRTMAC(PRTNUM,NAME);
 %DO I= 1 %TO &PRTNUM;
     PROC PRINT DATA=&NAME&I;
      TITLE "PRINT OF DATASET &NAME&I";
RUN;
 %END;
%MEND;

%PRTMAC(4,PAYROLL)
```

The Generated SAS Code

```
PROC PRINT DATA=PAYROLL1;
  TITLE "PRINT OF DATASET PAYROLL1";
RUN;
PROC PRINT DATA=PAYROLL2;
  TITLE "PRINT OF DATASET PAYROLL2";
RUN;
PROC PRINT DATA=PAYROLL3;
  TITLE "PRINT OF DATASET PAYROLL3";
RUN;
PROC PRINT DATA=PAYROLL4;
  TITLE "PRINT OF DATASET PAYROLL4";
RUN;
```

SAS DATA Step Interfaces

The SYMGET function and SYMPUT call routine can trasfer values between macro variables and the SAS Program Data Vector when the DATA step executes. This allows SAS steps to communicate to each other through macro variables and allows us to combine the power of the macro language, the DATA step, and the SAS procs together in our programs. Because the statements transfer values at EXECUTION time, macro variables created with SYMPUT, can be referenced via & in the NEXT step or later.

Example: Display the number of observations in a dataset in a title.

```
%MACRO OBSCOUNT(NAME);
  DATA _NULL_;
   SET &NAME NOBS=OBSOUT;
   CALL SYMPUT('MOBSOUT',OBSOUT);
   STOP;
  RUN;
  PROC PRINT DATA=&NAME;
   TITLE "DATASET &NAME CONTAINS &MOBSOUT
    OBSERVATIONS";
  RUN;
%MEND;

%OBSCOUNT(PAYROLL)
```

The Generated SAS Code

```
DATA _NULL_;
 SET PAYROLL NOBS=OBSOUT;
 CALL SYMPUT('MOBSOUT',OBSOUT);
 STOP;
RUN;
```

```
PROC PRINT DATA=PAYROLL;
 TITLE  "DATASET PAYROLL CONTAINS 50 OBSERVATIONS";
RUN;
```

A SAS Macro Application

Problem: Each day you read a SAS dataset containing data from counties in Wisconsin. Anywhere between 1 and 72 counties might report that day. Do the following:

1. Create a separate dataset for each reporting county.
2. Produce a separate PROC PRINT for each reporting county.
3. In the TITLE print the county name.
4. Reset the page number to 1 at the beginning of each report.
5. In a footnote print the number of observations processed for each county.

Question: How do you do it?
Solution: A Data Step and a SAS Macro.
A data step and a macro to generate the PROC PRINTs.

The data step goes through the data and:

- counts counties
- counts observations per county, puts in macro variables
- puts county names into macro variables
- puts total counties reporting into a macro variable.

The Data Step Code

```
DATA _NULL_;
SET COUNTYDT END=EOF;            /* READ SAS DATASET  */
 BY COUNTYNM;                    /* SORT SEQ          */
 IF FIRST.COUNTYNM THEN DO;      /* NEW COUNTY ?      */
    NUMCTY+1;                    /* ADD 1 TO NUMCTY   */
    CTYOBS=0;                    /* OBS PER CTY TO 0  */
    END;
 CTYOBS+1;                       /* ADD 1 OBS FOR CTY */
IF LAST.COUNTYNM THEN DO;        /* lst CTY, MK MACvar*/
    CALL SYMPUT('MCTY'||LEFT(PUT(NUMCTY,3.)),COUNTYNM);
    CALL
   SYMPUT('MOBS'||LEFT(PUT(NUMCTY,3.)),LEFT(CTYOBS));
    END;
IF EOF THEN
    CALL SYMPUT('MTOTCT',NUMCTY);/* tot no of ctys   */
RUN;
%PUT *** MTOTCT=&MTOTCT;          /* DISPLAY #OF CTYS */
```

If you prefer, PROC SQL can be used to produce the same results.

```
PROC SQL NOPRINT;
  SELECT  LEFT(PUT(COUNT(DISTINCT COUNTYNM),3.))
           INTO:MTOTCT FROM COUNTYDT;
                          /* NO OF UNIQUE CTYS     */
  SELECT  DISTINCT COUNTYNM /* EACH CTY NAME        */
          INTO:MCTY1-:MCTY&MTOTCT FROM COUNTYDT;
  SELECT  COUNT(*)          /* OBS PER              */
          INTO:MOBS1-:MOBS&MTOTCT FROM COUNTYDT
  GROUP BY COUNTYNM;

%PUT *** MTOTCT=&MTOTCT;    /* DISPLAY NO OF CTYS    */
```

We can now write a macro to loop through the macro variables and produce the necessary proc steps.

```
%MACRO COUNTYMC;                /* MACRO START       */
%DO I=1 %TO &MTOTCT;            /* LOOP THRU ALL CTYS */
 %PUT *** LOOP &I OF &MTOTCT;   /* DISPLAY PROGRESS  */
 PROC PRINT DATA=COUNTYDT;      /* PROC PRINT        */
  WHERE COUNTYNM="&&MCTY&I";    /* GENERATED WHERE   */
  OPTIONS PAGENO=1;             /* RESET PAGENO      */
  TITLE "REPORT FOR COUNTY &&MCTY&I";
                                /* TITLES & FOOTNOTES */
  FOOTNOTE "TOTAL OBSERVATION COUNT WAS &&MOBS&I";
 RUN;
%END;                           /* END OF %DO        */
%MEND COUNTYMC;                 /* END OF MACRO      */
%COUNTYMC                       /* INVOKE MACRO      */
```

The Generated Code

```
*** MTOTCT=3
*** LOOP 1 OF 3
   PROC PRINT DATA=COUNTYDT;
   WHERE COUNTYNM="ASHLAND";   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY ASHLAND";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS 2";
 RUN;
*** LOOP 2 OF 3
   PROC PRINT DATA=COUNTYDT;
   WHERE COUNTYNM="BAYFIELD";   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY BAYFIELD";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS 3";   RUN;
*** LOOP 3 OF 3
   PROC PRINT DATA=COUNTYDT;
   WHERE COUNTYNM="WASHINGTON";   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY WASHINGTON";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS 1";  RUN;
```

If a SAS user understands where all of the structures are stored in the above example, and when each activity occurs, a good understanding of macros exists. This type of application can be used as a template for lots of different SAS problems.

Example:
A very common problem is to run a proc against all members of a SAS library.

The following program produces a SAS dataset containing an observation for each variable within each dataset in the library . A PROC PRINT can display it's values.

```
 PROC CONTENTS DATA=WORK._ALL_ NOPRINT OUT=CONTOUT;
  RUN;
 PROC PRINT DATA=CONTOUT;
 VAR LIBNAME MEMNAME NAME;
 TITLE 'CONTOUT';
RUN;
```

Could we write a SAS macro that will generate a separate PROC PRINT of all members in the library with an appropriate title, noting that we want to produce one print per member, not one per variable?
Yes!

```
DATA ONEMEM;
 SET CONTOUT END=EOF;
  BY MEMNAME;
  IF LAST.MEMNAME;
  KTR+1;
  CALL SYMPUT ('MMEM'||LEFT(PUT(KTR,5.)),MEMNAME);
  IF EOF;
  CALL SYMPUT ('MTOTOBS',LEFT(PUT(KTR,5.)));
RUN;
%MACRO PRTLOOP;
 %DO I = 1 %TO &MTOTOBS;
     PROC PRINT DATA=&&MMEM&I;
        TITLE "&&MMEM&I";
     RUN;
 %END;
%MEND PRTLOOP;
%PRTLOOP
```

The SSCFLAT Macro

A general purpose macro that:

- converts any SAS ds to a comma delimited file for input to spreadsheets etc.
- will run on all platforms
- automatically reads dictionary tables to get ds definition
- honors SAS formatting
- can create a header line
- is available for the asking
- demonstrates many of the topics covered.

SSCFLAT Macro Partial Source

```
%MACRO SSCFLAT(MSASDS=,   /*INPUT SAS DS (REQUIRED  */
   MPREFIX=&SYSPREF..,    /*OUT PREF, OR DIR OUT    */
   MFLATOUT=&MPREFIX&MMEMNAME..DAT,  /*FLATFILE OUT*/
```

```
   MHEADER=YES,           /*FIELD NAMES IN FIRST REC
        */
   MLIST=YES,             /*PRINT FLAT FILE IN LOG?*/
   MTRIMNUM=YES,          /*TRIM NUM TRAIL BLANKS? */
   MTRACE=NO,            /*DEBUGGING OPTION       */
   MMISSING=".",          /*MISSING VALUE CHARACTER
        */
   MLRECL=6160,           /*LARGEST RECORD LENGTH  */
   MVSOPT=UNIT=3390,     /*MVS UNIT OPTIONS       */
   MSPACE=1,              /*MVS SPACE IN CYLS      */
   );
```

A SSCFLAT Macro Example
Convert a SAS file to a CSV file.

```
%INC 'insert file ref\sscflat.sas.';
%SSCFLAT(MSASDS=WORK.ADDRDATA,
    mprefix=c:\temp\)  *invoke macro;
```

Creates a file called c:\temp\addrdata.dat that is now available for download and/or import to spreadsheets, etc.
Anyone interested in the above macro, or with any questions or comments can contact us as shown below.

Conclusion

The SAS macro facility gives the SAS programmer the ability to introduce logic in the generation of dynamic SAS programs by utilizing this second programming language. The price for this feature is a more complex program with more detail to timing and structures required.

Contact Information

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

Steven First
Systems Seminar Consultants
2997 Yarmouth Greenway Drive
Madison, WI 53716
608 278-9964 x 302 voice
608 278-0065 fax
sfirst@sys-seminar.com
www.sys-seminar.com