



# YOUR SAS TECHNOLOGY REPORT

OCTOBER 2005

## Dear SAS Technology Report Readers,

SAS is committed to keeping our customers informed about our products. In fact, we've created a special Web site just for you. You can find out about technical support, documentation, training and much more at <http://support.sas.com>.

If you're interested in Base SAS or migration, for instance, visit the [Communities](#) section of the site. This area offers you an intimate look at new and existing SAS products and solutions.

And if you want to find a particular technical tip, visit [Samples](#). You'll be able to search through thousands of tech tips. I hope you'll find what you're looking for on our customer Web site. If not, please send me an e-mail at [techeditor@sas.com](mailto:techeditor@sas.com).

**Thanks for your interest in SAS!**

A handwritten signature in black ink that reads "Shelley Sessoms".

Shelley Sessoms  
Editor, *Your SAS Technology Report*

## Contact

We welcome feedback and article ideas.  
Send e-mail to [techeditor@sas.com](mailto:techeditor@sas.com).

## Give Your Macro Code an Extreme Makeover:

*Tips for even the most seasoned macro programmer*

By: Russ Tyndall

Many times when writing code we revert back to our old and comfortable ways. A lot of times these ways may be simplified and written more efficiently. In this paper I will demonstrate new techniques of writing programs by implementing new macro features that can replace longer, older programs. By the end of this paper you should be able to write programs a lot faster and with a lot less code than in previous versions.

### CALL SYMPUT VS. CALL SYMPUTX

Let's start by looking at common programming problems. In some situations you need to use CALL SYMPUT to create macro variables based on DATA step variables, but you want these macro variables to be global. The problem here is the uncertainty of how many macro variables are created, so you have to preprocess that data so that a %GLOBAL statement can be built, holding each of these variable names. Let's take a look at the old way of achieving this task versus the new and improved way.

The following code achieves the result described in the paragraph above:

```
data dusty;
  input dept $ name $ salary @@;
  cards;
bedding Watlee 18000   bedding Ives 16000
bedding Parker 9000   bedding George 8000
bedding Joiner 8000   carpet Keller 20000
carpet Ray 12000      carpet Jones 9000
gifts Johnston 8000   gifts Matthew 19000
kitchen White 8000   kitchen Banks 14000
kitchen Marks 9000   kitchen Cannon 15000
tv Jones 9000        tv Smith 8000
tv Rogers 15000      tv Morse 16000
;
```

#### Old Code:

```
%macro drive (class,var);
proc means noprint;
  class &class;
  var &var;
  output out=stats sum=s_sal;
run;
```

```

data _null_;
  set stats;
  if _n_=1 then call execute ('%global s_tot;');
  else call execute('%global s'||dept||;');
run;
data _null_;
  set stats;
  if _n_=1 then call symput('s_tot',trim(left(s_sal)));
  else call symput('s'||dept,trim(left(s_sal)));
run;
%mend drive;
%drive(dept,salary)
%put _user_;

```

Starting in SAS 9.0 you can use a new call routine, **CALL SYMPUTX**, to simplify this process. The four benefits of CALL SYMPUTX are:

- 1.) CALL SYMPUTX does not write a note to the SAS log when the second argument is numeric. CALL SYMPUT, however, writes a note to the log stating that numeric values were converted to character values.
- 2.) CALL SYMPUTX uses a field width of up to 32 characters when it converts a numeric second argument to a character value. CALL SYMPUT uses a field width of up to 12 characters.
- 3.) CALL SYMPUTX left-justifies both arguments and trims trailing blanks. CALL SYMPUT does not left-justify the arguments, and trims trailing blanks from the first argument only. Leading blanks in the value of name cause an error.
- 4.) CALL SYMPUTX enables you to specify the symbol table in which to store the macro variable, whereas CALL SYMPUT does not. This is accomplished by an optional third argument to the SYMPUTX function.

### **New Code:**

This program does the same as the one above except now we are using CALL SYMPUTX and you can see it took less code, as we were able to remove one of the data steps.

```

%macro drive(class,var);
proc means noprint;
  class &class;
  var &var;
  output out=stats sum=s_sal;
run;

data _null_;
  set stats;
  if _n_=1 then call symputx('s_tot',s_sal,'g');
  else call symputx('s'||dept,s_sal,'g');
run;
%mend drive;

```

```
%drive(dept,salary)
```

```
%put _user_;
```

### LOG:

```

1  data dusty;
2  input dept $ name $ salary @@;
3  cards;

```

NOTE: SAS went to a new line when INPUT statement reached past the end of a line.

NOTE: The data set WORK.DUSTY has 18 observations and 3 variables.

NOTE: DATA statement used (Total process time):

real time	0.03 seconds
cpu time	0.03 seconds

```

13 ;
14
15 %macro drive(class,var);
16 proc means noprint;
17   class &class;
18   var &var;
19   output out=stats sum=s_sal;
20 run;
21
22 data _null_;
23   set stats;
24   if _n_=1 then call symputx('s_tot',s_sal,'g');
25   else call symputx('s'||dept,s_sal,'g');
26 run;

```

```
27 %mend drive;
28
29 %drive(dept,salary)
```

NOTE: There were 18 observations read from the data set WORK.DUSTY.

NOTE: The data set WORK.STATS has 6 observations and 4 variables.

NOTE: PROCEDURE MEANS used (Total process time):

```
real time    0.03 seconds
cpu time     0.03 seconds
```

NOTE: There were 6 observations read from the data set WORK.STATS.

NOTE: DATA statement used (Total process time):

```
real time    0.01 seconds
cpu time     0.00 seconds
```

```
30
31 %put _user_;
GLOBAL S_TOT 221000
GLOBAL SKITCHEN 46000
GLOBAL SCARPET 41000
GLOBAL STV 48000
GLOBAL SGIFTS 27000
GLOBAL SBEDDING 59000
```

## **%SYSFUNC**

One of the most exciting things to happen within the macro facility was the introduction of the function: %SYSFUNC. This opened up a new and exciting way to code within macro. Tasks that took many steps to achieve can now be accomplished with fewer lines of code. %SYSFUNC allows you to use many SAS functions within the macro facility. These next few examples illustrate how this function can save time in writing macro programs.

### **Example 1:**

Before %SYSFUNC a DATA step was required just to place the number of observations in a SAS data set into a macro variable. The example below illustrates the old method:

#### **Old Code:**

```
%macro numobs(dsn);
%global num;
```

```

data _null_;
  if 0 then set &dsn nobs=count;
  call symput('num',left(put(count,8.)));
  stop;
run;
%mend numobs;

```

```
%numobs(dataset_name)
```

This will create a macro variable NUM that holds the number of observations in the data set.

Now with the use of the function %SYSFUNC along with the new ATTRN function the DATA step code can be eliminated from the macro and substituted with just these 3 lines of code:

### **New Code:**

```

%macro numobs(dsn);
  %global num;
  %let dsid=%sysfunc(open(&dsn));
  %let num=%sysfunc(attrn(&dsid,nobs));
  %let rc=%sysfunc(close(&dsid));
%mend numobs;

```

```
%numobs(dataset_name)
```

The first %LET statement opens the data set. The second %LET uses the ATTRN function to check the number of observations via the NOBS attribute. The third %LET statement closes the data set.

### **Example 2:**

A dramatic example of the value of %SYSFUNC is the code to check for the existence of a SAS data set. The old way of accomplishing this task is shown below:

### **Old Code:**

```

%macro exist(dsn);
  %global exist;
  %if &dsn ne %then %str(
  data _null_;
    if 0 then set &dsn;
    stop;
  run;

```

```

);
%if &syserr=0 %then %let exist=1;
%else %let exist=0;
%mend exist;

```

```
%exist(dataset_name_)
```

Now with the use of %SYSFUNC, along with the new EXIST function, the code above can be replaced with this single line of code:

### **New Code:**

```
%let exist=%sysfunc(exist(dataset_name));
```

Macro variable EXIST will be set to 1 if the data set exists and 0 if it does not exist.

### **Example 3:**

Another common task is to place each DATA step variable into a separate macro variable. The most common way to accomplish this task requires a PROC CONTENTS and a DATA step:

### **Old Code:**

```

%macro test;
proc contents data=sasuser.class out=new(keep=name);
run;
data _null_;
set new;
call symput('name'||trim(left(_n_)),trim(left(name)));
run;
%put _user_;
%mend test;

```

```
%test
```

With the use of %SYSFUNC you can eliminate the PROC CONTENTS and the DATA step code and retrieve the same information by using macro logic only.

```

%macro test;
%let dsid=%sysfunc(open(sasuser.class));
%let cnt=%sysfunc(attrn(&dsid,nvars));
%do i = 1 %to &cnt;
%let name&i=%sysfunc(varname(&dsid,&i));
%end;

```

```
%let rc=%sysfunc(close(&dsid));
%put _user_;
%mend test;
%test
```

The first %LET statement opens the data set. The second %LET uses the ATTRN function to check the number of variables via the NVAR attribute. The %DO statement is used to loop through the number of variables and placing each DATA step variable in its own macro variable via the %LET statement. The fourth %LET statement closes the data set.

#### **Example 4:**

Suppose you wanted to do something as simple as placing the current date and time within a title.

#### **Old Code:**

```
data _null_;
  call symput('date',put(today(),date.));
  call symput('time',put(time(),time.));
run;

proc print;
  title "Today is &date and the time is &time.";
run;
```

Now with %SYSFUNC this task is much easier because the function can be included in the TITLE statement. For example:

#### **New Code:**

```
proc print;
  title "Today is %sysfunc(today(),date.) and the time is %sysfunc(time(),time.)";
run;
```

## **NEW FEATURES PART A**

In this next section I want to discuss some new macro features that can simplify the way you may have written code in the past.

#### **Example 1: (%SYMDEL)**

Let's say we want to delete all global macro variables. The old way of accomplishing this task would not delete the macro variables, but it would set them to blank values.

### Old Code:

```
%let x=1;
%let y=2;
%put _user_;
%macro lst;
%global list;
%let list=;
data _null_;
  set sashelp.vmacro;
  if scope = 'GLOBAL' then do;
    call symput('mac',trim(left(name)));
    call execute('%let &mac=;');
  end;
run;
%mend;
%lst
%put _user_;
```

### LOG:

NOTE: SAS initialization used:

real time	1.56 seconds
cpu time	0.28 seconds

```
1  %let x=1;
2  %let y=2;
3  %put _user_;
GLOBAL X 1
GLOBAL Y 2
4  %macro lst;
5  %global list;
6  %let list=;
7  data _null_;
8  set sashelp.vmacro;
9  if scope = 'GLOBAL' then do;
10 call symput('mac',trim(left(name)));
11 call execute('%let &mac=;');
12 end;
13 run;
14 %mend;
15 %lst
```

NOTE: DATA statement used:  
real time 0.01 seconds  
cpu time 0.00 seconds

NOTE: There were 49 observations read from the data set SASHELP.VMACRO.

NOTE: CALL EXECUTE routine executed successfully, but no SAS statements were generated.

```
16  
17 %put _user_;  
GLOBAL X  
GLOBAL Y  
GLOBAL LIST  
GLOBAL MAC
```

Notice in the results above, the macro variables still exist, but they are now set to null.

Starting with SAS 8.2 you can use the new macro statement **%SYMDEL**, to delete macro variables. The code below modifies the previous example by using %SYMDEL. The CALL EXECUTE allows us to execute the %SYMDEL for every observation for which the IF condition is true.

#### **New Code:**

```
%let x=1;  
%let y=2;  
%put _user_;  
%macro delvars;  
data vars;  
set sashelp.vmacro;  
run;  
data _null_;  
set vars;  
if scope='GLOBAL' then  
call execute('%symdel '||trim(left(name))||');  
run;  
%mend;  
%delvars
```

```
%put _user_;
```

#### **LOG:**

NOTE: SAS initialization used:  
real time 0.93 seconds  
cpu time 0.37 seconds

```

1  %let x=1;
2  %let y=2;
3  %put _user_;
GLOBAL X 1
GLOBAL Y 2
4  %macro delvars;
5    data vars;
6      set sashelp.vmacro;
7    run;

8  data _null_;
9    set vars;
10   if scope='GLOBAL' then
11     call execute('%symdel '||trim(left(name))||');
12   run;
13 %mend;
14 %delvars

```

NOTE: There were 47 observations read from the data set SASHELP.VMACRO.

NOTE: The data set WORK.VARS has 47 observations and 4 variables.

NOTE: DATA statement used:

real time	0.04 seconds
cpu time	0.01 seconds

NOTE: DATA statement used:

real time	0.01 seconds
cpu time	0.01 seconds

NOTE: There were 47 observations read from the data set WORK.VARS.

NOTE: CALL EXECUTE routine executed successfully, but no SAS statements were generated.

```

15
16 %put _user_;

```

As you can see the macro variables have been completely removed from the symbol table.

There is also a NOWARN option that can be placed on the %SYMDEL statement, for cases where you may try to delete a macro variable that does not exist. For example:

### **SAMPLE CODE:**

```
%symdel x;
```

```
%symdel x / nowarn;
```

### **LOG:**

WARNING: Attempt to delete macro variable X failed. Variable not found.

```
17 %symdel x;  
18  
19 %symdel x / nowarn;
```

Notice that you referenced a macro variable X that does not exist. The first SYMDEL, without the option, gives you a warning. The second statement, with the option NOWARN, gives no warning. There is also a DATA step call routine equivalent to %SYMDEL called CALL SYMDEL.

### **Example 2: (%SYMEXIST)**

The following example illustrates the code needed to check if a macro variable exists without getting a warning in the log.

```
%macro check(mvar);  
  %local i tmp;  
  %let dsid=%sysfunc(open(sashelp.vmacro));  
  %let num=%sysfunc(varnum(&dsid,name));  
  %do %until(&ob = -1);  
    %let i=%eval(&i+1);  
    %let ob=%sysfunc(fetchobs(&dsid,&i));  
    %let val=%sysfunc(getvarc(&dsid,&num));  
  
    %if &val = %upcase(&mvar) %then %do;  
      %let ob = -1;  
      %let tmp=1;  
    %end;  
  
    %else %do;  
      %let tmp=0;  
    %end;  
  
    %if &ob=-1 %then %do;  
      &tmp  
    %end;  
  
  %end;  
  %let rc=%sysfunc(close(&dsid));  
%mend check;
```

```
/** Check for the existence of the macro variable abc */  
%put Return 1 if macro variable exist:%check(abc);
```

Beginning with SAS 9.1, the **%SYMEXIST** function can be used to determine if a macro variable exists. The function returns a value of 1 if the macro variable exists and a value of 0 if it does not. The macro above can be replaced with this single line of code:

```
%put Return 1 if macro variable exist: %symexist(abc);
```

Note: The & (ampersand) does not precede the macro variable name in the argument to the function. There is also a DATA step function equivalent to %SYMEXIST called SYMEXIST.

### **Example 3: (%SYMGLOBL)**

To take the example above a step further, what if you wanted to check to see if a macro variable was declared as global. The code below would accomplish this task:

```
%let x=1;  
%macro check(name);  
  %global ans;  
  data _null_;  
    set sashelp.vmacro;  
    if upcase(scope)='GLOBAL' and name="%upcase(&name)" then do;  
      call symput('ans',trim(left(1)));  
      stop;  
    end;  
    else call symput('ans',trim(left(0)));  
  run;  
%mend check;  
%check(x);  
%put 1 if macro variable is global: &ans;
```

Starting in SAS 9.1 there is a new macro function called **%SYMGLOBL** which returns a 1 if the macro variable is global in scope or 0 otherwise. So the entire program above could be rewritten as:

```
%put 1 if macro variable is global: %symglobl(x);
```

There is also a similar function called **%SYMLOCAL** that checks for a macro variable on the local symbol table.

There are also DATA step functions equivalent to %SYMGLOBL and %SYMLOCAL called SYMGLOBL and SYMLOCAL.

#### **Example 4: (SAS\_EXECFILENAME)**

Another popular question is: how can I retrieve the name of the program that is currently executing?

This is a simple process running in batch. Using %SYSFUNC in conjunction with the GETOPTION function makes it easy. Consider the following:

```
%put The current program is %sysfunc(getoption(sysin));
```

Since SYSIN is used for batch mode source files it requires a little more effort to retrieve this information interactively. The following code can be run interactively to retrieve the path and the name of the current program:

```
%macro pname;
%global pgmname;
%let pgmname=;
data _null_;
set sashelp.vextfl;
if (substr(fileref,1,3)='_LN' or
    substr(fileref,1,3)='#LN' or
    substr(fileref,1,3)='SYS') and
index(upcase(xpath),'.SAS')>0 then do;
call symput("pgmname",trim(xpath));
stop;
end;
run;
%mend pname;
%pname;
%put pgmname=&pgmname;
```

Beginning in SAS 9.0 the macro above is not needed. There is a new environment variable for the Enhanced Editor called: **SAS\_EXECFILENAME**. This does not include the full path, only the filename.

The following will now return the executing program when running interactively:

```
%put %sysget(SAS_EXECFILENAME);
```

In order to get the full path, there is also an environment variable for the Enhanced Editor called **SAS\_EXECFILEPATH** that contains the full path of the submitted program or catalog entry. The full path includes the folder and the filename.

## NEW FEATURES PART B

This section discusses new macro features which extend the existing capabilities of the SAS System. You may find these features beneficial when programming within macro.

### Example 1: (LIBRARY CONCATENATION)

Unlike formats and the FMTSEARCH option, prior to SAS 7 there is no method to concatenate Stored Compiled Libraries.

Starting in SAS 7 you have the ability to implicitly concatenate SAS catalogs. Because the LIBNAME statement allows you to logically concatenate SAS data libraries, SAS catalogs that have the same name are also implicitly concatenated. All Stored Compiled macros are stored in a catalog called SASMACR. This feature enables you to reference multiple SASMACR catalogs in different SAS data libraries that have been stored in the SASMSTORE= option. SAS searches for the SASMACR catalog that is located in each library that is referenced in the LIBNAME statement and specified in the SASMSTORE= option until the entry (invoked macro) is found.

#### Syntax:

```
LIBNAME libref <engine> (library-specification-1 <...library-specification-  
n>)<options>;
```

If the library specification is a path, then each path must be enclosed in single quotation marks. Here is an example:

```
libname allmine ('c:\test1' 'c:\test2' 'c:\test3');  
options sasmstore=allmine mstored;
```

Suppose you were invoking a macro called TEST and was using the LIBNAME and OPTIONS statement above. SAS searches the SASMACR catalog in c:\test1, then c:\test2, etc. and uses the first one that is found.

### Example 2: (CAPTURING LOG MESSAGES)

What if you have a situation where you want to grab the text of the last error/warning message from the log and place this in a variable. You have to process the log file, read

in the text, locate the last error and store this in a macro variable. In this example, errmess contains the last error message:

```
%let subdir=c:\errors\;
filename dir pipe "dir &subdir.*.log /B";

data new;
  infile dir trunccover;
  input filename $80.;
  filename="&subdir" || filename;
  length lname logfile $30;
  infile dummy filevar=filename filename=lname end=done trunccover;
  do while (not done);
    input test $5. @;
    if test='ERROR' then do;
      input @6 msg $76.;
      logfile=lname;
    end;
    else input;
  end;
  call symput('errmess',trim(left(msg)));
run;
```

The log was saved to a directory called errors with the file having a .log extension. The first INPUT statement reads a log file from the subdirectory. The second INPUT statement reads a record from that log file searching for the word ERROR. If ERROR is found the third INPUT is used to create the data set variable called msg that contains the error message. This is later placed in a macro variable called errmess via the CALL SYMPUT.

This task is much easier starting in SAS 9.2 with two new automatic macro variables entitled: **SYSERRORTTEXT** and **SYSWARNINGTEXT**. **SYSERRORTTEXT** contains the text of the last error message generated within the SAS log. **SYSWARNINGTEXT** contains the text of the last warning message generated within the SAS log. This one line replaces the code above:

```
%let errmess=&syserrorttext;
```

Here is an example using these two automatic macro variables.

### **SAMPLE CODE:**

```
data NULL;
  set doesnotexist;
```

run;

%put &syserrortext;  
%put &syswarningtext;

**LOG:**

1 data NULL;  
2 set doesnotexist;

ERROR: File WORK.DOESNOTEXIST.DATA does not exist.

3 run;

NOTE: The SAS System stopped processing this step because of errors.

WARNING: The data set WORK.NULL may be incomplete. When this step was stopped there were 0 observations and 0 variables.

NOTE: DATA statement used (Total process time):

real time 11.16 seconds

cpu time 0.07 seconds

4 %put &syserrortext;

File WORK.DOESNOTEXIST.DATA does not exist.

5 %put &syswarningtext;

The data set WORK.NULL may be incomplete. When this step was stopped there were 0 observations and 0 variables.

**Example 3: (SECURE)**

In previous releases of SAS you had no way to truly hide code within a compiled macro. This is no longer the case starting in SAS 9.2 with the introduction of a new %MACRO statement option called **SECURE**.

This option causes the contents of a macro to be encrypted when stored in a stored compiled macro library. This feature enables you to write secure macros that will protect intellectual property that is contained in the macros. This is done using the Encryption Algorithm Manager. The SECURE option can only be used in conjunction with the STORE option.

A NOSECURE option has been implemented to aid in doing a global edit of a source file or library to turn on security. If you are writing several macros that will need to be encrypted, you can create them using the NOSECURE option. When all macros are

completed and ready for production, you can do a global edit and change NOSECURE to SECURE.

The following example shows using the STORE and SECURE options to create a macro that is encrypted.

```
Libname mylib 'c:\mymac';
options mstored sasmstore=mylib;
%macro secure/store secure;
  data _null_;
    x=1;
    put "This data step was generated from a secure macro.";
  run;
%mend secure;
%secure
filename maccat catalog 'mylib.sasmacr.secure.macro';
data _null_;
  infile maccat;
  input;
  list;
run;
```

After executing the above code this compiled macro is encrypted and its contents cannot be viewed.

#### **Example 4: (SOURCE and %COPY)**

In previous releases of SAS, you were unable to retrieve the source code from a compiled macro. Starting in SAS 9.1, **SOURCE**, a new option exists for the %MACRO statement that, when used with the existing STORE option combines and stores the source of the compiled macro. The compiled macro code becomes an entry in a SAS catalog in a permanent SAS data library. The compiled macro and the source code are stored together in the same SASMACR catalog.

The SOURCE option requires that the STORE option and the SAS option MSTORED be set. You can use the SAS option SASMSTORE= to identify a permanent SAS data library. You can store a macro or call a stored compiled macro only when the SAS option MSTORED is in effect.

**Note:** The source code saved by the SOURCE option begins with the %MACRO keyword and ends with the semi-colon following the %MEND statement.

Now that you have a way to store the source code with the SOURCE option, you also need a way to retrieve this information. The answer is the new %COPY statement, which copies specified items from a SAS macro library.

Syntax:

`%COPY Macro-name </options(s)>`

*Macro-name*

name of the macro that the %COPY statement will use.

option(s)

can be one or more of the following options:

`LIBRARY= <libref>`

`LIB=`

specifies the libref of a SAS data library that contains a catalog of stored compiled SAS macros. If no library is specified, the libref specified by the SASMSTORE= option is used. Restriction: This libref cannot be WORK.

`OUTFILE=<fileref> | <'external file'>`

`OUT=`

specifies the output destination of the %COPY statement. The value can be a fileref or an external file.

`SOURCE`

`SRC`

specifies that the source code of the macro will be copied to the output destination. If the OUTFILE= option is not specified, the source is written to the SAS log.

Below is an example using the new SOURCE option along with the %COPY statement:

**SAMPLE CODE:**

```
libname test 'c:\';
```

```
options mstored sasstore=test;
```

```
%macro test(arg) / store source des="test of the source option";
```

```
  %put arg = &arg;
```

```
  data one;
```

```
    x=1;
```

```
  run;
```

```
%mend test;
```

```
%copy test / source;
```

**LOG:**

```
149 libname test 'c:\';  
NOTE: Libref TEST was successfully assigned as follows:  
    Engine:      V9  
    Physical Name: c:\  
150 options mstored sasmstore=test;  
151  
152 %macro test(arg) / store source des="test of the source option";  
153 %put arg = &arg;  
154 data one;  
155 x=1;  
156 run;  
157 %mend test;  
158  
159 %copy test / source;  
%macro test(arg) / store source des="test of the source option";  
%put arg = &arg;  
data one;  
x=1;  
run;  
%mend test;
```

**Limitations:**

The SOURCE option cannot be used on nested macro definitions (macro definitions contained within another macro).

If the SECURE option is used in conjunction with the SOURCE option, the SECURE option takes precedence and the source cannot be viewed due to encryption.

**Limitations continued:**

Macro catalogs cannot be moved across different releases of SAS or across different platforms.

**Additional Options, Statements, and Automatic Macro Variables**

This final section contains other new macro features that you may also find helpful when coding within the macro facility.

**New option for %MACRO statement:**

**MINDELIMITER:** Assigns the value to be used as the delimiter for the IN (#) operator within the macro at macro compilation (starting in SAS 9.2)

Syntax:

```
%MACRO <macro-name> ( <arguments > ) / MINDELIMITER='<single character>';
```

The value of the /MINDELIMITER option must be a single character enclosed in single quotation marks. The value specified by the /MINDELIMITER option overrides the value of the MINDELIMITER= global option. The default value is a blank. The /MINDELIMITER= option may appear only once in a %MACRO statement.

### **SAMPLE CODE:**

```
%macro test / mindelimiter='';  
  %if &x in (1,2,3) %then %put this is true;  
  %else %put this is false;  
%mend test;  
%let x=3;  
%test
```

### **LOG:**

```
1 %macro test / mindelimiter='';  
2 %if &x in (1,2,3) %then %put this is true;  
3 %else %put this is false;  
4 %mend test;  
5 %let x=3;  
6 %test
```

this is true

### **New Automatic macro variables:**

**&SYSMACRONAME :** Returns the name of the currently executing macro (started in SAS 8.2)

Here is an example of using SYSMACRONAME:

### **SAMPLE CODE:**

```
%macro test2;  
  %put &sysmacroname is executing;  
  %put this is another test;  
%mend test2;
```

```
%macro test;
  %put this is a test;
  %test2
%mend test;
%test
```

**LOG:**

```
103 %macro test2;
104 %put &sysmacroname is executing;
105 %put this is another test;
106 %mend test2;
107 %macro test;
108 %put this is a test;
109 %test2
110 %mend test;
111 %test
this is a test
TEST2 is executing
this is another test
```

**&SYSPROCNAME:** Contains the name of the procedure (or DATASTEP for DATA steps) currently being processed by the SAS Language Processor (started in SAS 8.2). This must be checked between the procedure statement and the next step boundary.

**SAMPLE CODE:**

```
proc print data=sashelp.class;
  %put &sysprocname;
run;
```

```
data one;
  x=100;
  put "&sysprocname";
run;
```

**LOG:**

```
143 proc print data=sashelp.class;
144 %put &sysprocname;
PRINT
145 run;
```

NOTE: There were 19 observations read from the data set SASHELP.CLASS.

NOTE: PROCEDURE PRINT used (Total process time):

real time	0.01 seconds
cpu time	0.01 seconds

```
146
147 data one;
148 x=100;
149 put "&sysprocname";
150 run;
```

#### DATASTEP

NOTE: The data set WORK.ONE has 1 observations and 1 variables.

NOTE: DATA statement used (Total process time):

real time	0.03 seconds
cpu time	0.01 seconds

**&SYSNCPU:** Contains the current number of processors available to SAS for computations (started in SAS 9.0)

This macro variable was created to help determine whether you are taking advantage of the new parallel-processing abilities in SAS, &SYSNCPU contains the current number of CPUs that SAS can use during the current SAS session.

&SYSNCPU is an automatic macro variable that provides the current value of the CPUCOUNT option. For more information about CPUCOUNT system option, see the SAS Language Reference: Dictionary.

**&SYSENCODING:** Contains the name of the current session encoding (starting in SAS 9.2)

The following statement displays the encoding for the SAS session.

```
%put The encoding for this SAS session is: &sysencoding;
```

Executing this statement writes the following to the SAS log:

```
The encoding for this SAS session is: wlatin1
```

Note: The value 'wlatin1' may be different on your OS depending on the encoding you are using.

#### **New macro system options for debugging:**

**MEEXECNOTE:** Displays macro execution information in the SAS log at macro invocation. The default value is NOMEXECNOTE. (started in SAS 9.0)

## **SAMPLE CODE:**

```
options mexecnote;
```

```
%macro test;  
  %do i = 1 %to 3;  
    %put &i;  
  %end;  
%mend test;  
%test
```

## **LOG:**

```
2  options mexecnote;  
3  
4  %macro test;  
5    %do i = 1 %to 3;  
6      %put &i;  
7    %end;  
8  %mend test;  
9  %test
```

NOTE: The macro TEST is executing from memory.

```
1  
2  
3
```

**MCOMPILENOTE:** Specifies that a NOTE be issued to the SAS log when the compilation of a macro is completed and also writes out the size and number of instructions upon the completion of the compilation of any macro. MCOMPILENOTE is set to none by default. (started in SAS 9.0)

Syntax:

```
MCOMPILENOTE=<NONE | NOAUTOCALL | ALL>
```

**NONE**

prevents any NOTE from being written to the log.

**NOAUTOCALL**

prevents any NOTE from being written to the log for AUTOCALL macros, but does issue a NOTE to the log upon the completion of the compilation of any other macro.

**ALL**

Print compilation about all macros that are being compiled

**MPRINTNEST:** Enables the macro nesting information to be displayed in the MPRINT output in the SAS log. Default value is NOMPRINTNEST. (started in SAS 9.0)

This has no effect on the MPRINT output that is sent to an external file. For more information, see the MFILE system option.

The setting of MPRINTNEST does not imply the setting of MPRINT. You must set both MPRINT and MPRINTNEST in order for output (with the nesting information) to be written to the SAS log.

**SAMPLE CODE:**

```
%macro outer;

data _null_;
  %inner
run;

%mend outer;
%macro inner;
  put %inrmst;
%mend inner;
%macro inrmst;
  'This is the text of the PUT statement'
%mend inrmst;

options mprint mprintnest;
%outer
```

**LOG:**

```
MPRINT(OUTER): data _null_;
MPRINT(OUTER.INNER): put
MPRINT(OUTER.INNER.INRMOST): 'This is the text of the PUT statement'
MPRINT(OUTER.INNER): ;
MPRINT(OUTER): run;
This is the text of the PUT statement
```

NOTE: DATA statement used (Total process time):

real time	0.10 seconds
cpu time	0.06 seconds

**MLOGICNEST:** Enables the macro nesting information to be displayed in the MLOGIC output in the SAS log. Default value is NOMLOGICNEST. (started in SAS 9.0)

The setting of MLOGICNEST does not affect the output of any currently executing macro.

The setting of MLOGICNEST does not imply the setting of MLOGIC. You must set both MLOGIC and MLOGICNEST in order for output (with nesting information) to be written to the SAS log.

**SAMPLE CODE:**

```
%macro outer;
  %put THIS IS OUTER;
  %inner;
%mend outer;
%macro inner;
  %put THIS IS INNER;
  %inrmmost;
%mend inner;
%macro inrmmost;
  %put THIS IS INRMOST;
%mend;
options mlogic mlogicnest;
%outer
```

**LOG:**

```
MLOGIC(OUTER): Beginning execution.
MLOGIC(OUTER): %PUT THIS IS OUTER
THIS IS OUTER
MLOGIC(OUTER.INNER): Beginning execution.
MLOGIC(OUTER.INNER): %PUT THIS IS INNER
THIS IS INNER
MLOGIC(OUTER.INNER.INRMOST): Beginning execution.
MLOGIC(OUTER.INNER.INRMOST): %PUT THIS IS INRMOST
THIS IS INRMOST
MLOGIC(OUTER.INNER.INRMOST): Ending execution.
MLOGIC(OUTER.INNER): Ending execution.
MLOGIC(OUTER): Ending execution.
```

**MAUTOLOCDISPLAY:** Specifies that the source location of the autocall macro be displayed in the SAS log when the autocall macro is invoked. Default value is NOMAUTOLOCDISPLAY. (started in SAS 9.0)

**SAMPLE CODE:**

```
options mautilocdisplay;
```

```
%let x=%trim(abc);
```

**LOG:**

```
26 options mautilocdisplay;
```

```
27
```

```
28 %let x=%trim(abc);
```

```
MAUTOLOCDISPLAY(TRIM): This macro was compiled from the autocall file  
C:\SASv9\sas\dev\mva-v920\shell\auto\en\trim.sas
```

**New macro system options for operational needs:**

**MEXECSIZE:** Specifies the maximum size macro to be executed in memory. Default value is 65536. (starting in SAS 9.2)

Syntax:

MEXECSIZE=*n* | *nK* | *nM* | *nG* | *nT* | *hexX* | MIN | MAX

*n* specifies the maximum size macro to be executed in memory available in bytes.

*nK* specifies the maximum size macro to be executed in memory available in kilobytes.

*nM* specifies the maximum size macro to be executed in memory available in megabytes.

*nG* specifies the maximum size macro to be executed in memory available in gigabytes.

*nT* specifies the maximum size macro to be executed in memory available in terabytes.

MIN specifies the minimum size macro to be executed in memory. Minimum value is 0.

MAX specifies the maximum size macro to be executed in memory. Maximum value is 2,147,483,647.

*hexX* specifies the maximum size macro to be executed in memory by a hexadecimal number followed by an X.

Use the MEXECSIZE option to control the maximum size macro that will be executed in memory as opposed to being executed from a file. The MEXECSIZE option value refers to the compiled size of the macro. Memory is only allocated when the macro is executed. Once the macro completes, the memory is released. If memory is not available to execute the macro, an out of memory message is written to the SAS log. Use the MCOMPILENOTE option to write to the SAS log the size of the compiled macro. The MEMSIZE option has no relation to the MEXECSIZE option.

**MINOPERATOR:** Recognize special operators (IN, #) when evaluating logical or integer expressions. Default setting is NOMINOPERATOR. (starting in SAS 9.2).

**NOMINOPERATOR:** Do not recognize special infix operators when evaluating logical or integer expressions. This option is useful for jobs where IN needs to be treated as text rather than a special operator. (starting in SAS 9.2)

**MINDELIMITER:** Specifies the character to be used as the delimiter for the macro IN operator. Default value is a blank. (starting in SAS 9.2)

Syntax:

```
MINDELIMITER=<"option">
```

option:

is a character enclosed in double or single quotation marks. The character will be used as the delimiter for the macro IN operator.

**Example:**

```
options mindelimiter='';
```

The option value is retained in original case and can have a maximum length of one character. The default value of the MINDELIMITER option is a blank.

You can use the # character instead of IN.

**Note:** When the IN or # operator is used in a macro, the delimiter that is used at the execution time of the macro is the value of the MINDELIMITER option at the time of the compilation of the macro. The option MINOPERATOR must be on to use the IN operator starting in SAS 9.2. The setting of MINDELIMITER does not imply the setting of MINOPERATOR.

**SAMPLE CODE:**

```
%put %eval(a in d,e,f,a,b,c);
```

```
%put %eval(a in d e f a b c);
```

```
option mindelimiter='';  
%put %eval(a in d,e,f,a,b,c);
```

```
%put %eval(a in d e f a b c);
```

Notice in the log below how setting the value of MINDELIMITER affects the outcome.

**LOG:**

```
NOTE: SAS initialization used:  
      real time      1.02 seconds
```

```

cpu time      0.63 seconds

%put %eval(a in d,e,f,a,b,c);

0

%put %eval(a in d e f a b c);

1

option mindelimiter='';
%put %eval(a in d,e,f,a,b,c);

1

%put %eval(a in d e f a b c);
0

```

### **New Macro Statements:**

**%ABORT:** Stops the macro that is executing along with the current DATA step, SAS job, or SAS session (started in SAS 9.1)

Syntax :

```
%ABORT <ABEND | RETURN <n>> ;
```

**%RETURN:** Causes normal termination of the currently executing macro (started in SAS 9.1)

**Comparison:** The %ABORT and %RETURN statements are similar to the ABORT and RETURN statements found in the DATA step.

### **SAMPLE CODE:**

In the sample code below, if the error variable is set to 1, then the macro will stop executing and the DATA step will not execute.

```

%macro checkit(error);
  %if &error = 1 %then %return;

  data a;
    x=1;
  run;

```

```
%mend checkit;
```

```
%checkit(0)
```

```
%checkit(1)
```

**LOG:**

```
29 %macro checkit(error);  
30   %if &error = 1 %then %return;  
31  
32   data a;  
33     x=1;  
34   run;  
35  
36 %mend checkit;  
37  
38 %checkit(0)
```

NOTE: The data set WORK.A has 1 observations and 1 variables.

NOTE: DATA statement used (Total process time):

real time	1.01 seconds
cpu time	0.12 seconds

```
39 %checkit(1)
```

For complete documentation on new 9.2 features, see the 9.2 OnlineDoc when it becomes available.

See next page for a quick reference of all these new features.

Here is a quick reference of all the new macro features with their definition and the release of SAS in which they were introduced:

### **New macro feature for SAS 6.12**

-----

`%SYSFUNC` -- macro function to execute SAS functions or user-written functions.

### **New macro features for SAS 8.2**

-----

`%SYMDEL` -- macro statement that deletes the specified variables(s) from the macro global symbol table.

`&SYSMACRONAME` -- automatic macro variable that returns the name of the currently executing macro.

`&SYSPROCNAME` -- automatic macro variable that contains the name of the procedure (or `DATAS`TEP for `DATA` steps) currently being processed by the SAS Language Processor.

### **New features for SAS 9.0**

-----

`CALL SYMPUTX` -- `DATA` step call routine that assigns a value to a macro variable and removes both leading and trailing blanks, also allows you to select the symbol table where the macro variable will be placed.

`SAS_EXECFILENAME` -- environment variable for the Enhanced Editor that contains only the name of the submitted program or the catalog entry name.

`SAS_EXECFILEPATH` -- environment variable for the Enhanced Editor that contains the full path of the submitted program or catalog entry. The full path includes the folder and the filename.

`&SYSNCPU` -- automatic macro variable that contains the current number of processors available to SAS for computations.

`MEXECNOTE` -- system option that specifies whether to display macro execution information in the SAS log at macro invocation.

`MCOMPILENOTE=< ALL | NOAUTOCALL | NONE >` -- system option that issues a `NOTE` to the SAS log upon the completion of the compilation of a macro.

`MPRINTNEST` -- system option that allows the macro nesting information to be displayed in the `MPRINT` output in the SAS log.

MLOGICNEST -- system option that allows the macro nesting information to be displayed in the MLOGIC output in the SAS log.

MAUTOLOCDISPLAY -- system option that displays the source location of the autocall macros in the SAS log when the autocall macro is invoked.

### **New macro features for SAS 9.1**

-----

%SYMEXIST -- macro function that returns an indication as to whether the named macro variable exists.

%SYMGLOBL -- macro function that returns an indication as to whether the named macro variable is global in scope.

### **New macro features for SAS 9.1 continued**

-----

%SYMLOCAL -- macro function that returns an indication as to whether the named macro variable is local in scope.

SOURCE -- %MACRO statement option that combines and stores the source of the compiled macro with the compiled macro code as an entry in a SAS catalog in a permanent SAS data library.

%COPY -- macro statement that copies specified items from a SAS library.

%ABORT < ABEND | RETURN > < n > -- macro statement that stops the macro that is executing along with the current DATA step, SAS job, or SAS session.

%RETURN -- macro statement that causes normal termination of the currently executing macro.

### **New macro features for SAS 9.2**

-----

MINDELIMITER= -- system option that specifies the character to be used as the delimiter for the macro IN operator.

MINOPERATOR -- Recognize special infix operators when evaluating logical or integer expressions.

NOMINOPERATOR-- Do not recognize special infix operators when evaluating logical or integer expressions.

**MINDELIMITER=** -- %MACRO statement option that specifies a value that will override the value of the **MINDELIMITER=** global option.

**&SYSERRORTXT** -- automatic macro variable that contains the text of the last error message formatted for display on the SAS log.

**&SYSWARNINGTEXT** -- automatic macro variable that contains the text of the last warning message formatted for display on the SAS log.

**SECURE** -- %MACRO statement option that enables you to write secure macros which will protect intellectual property contained in stored compiled macros.

**&SYSENCODING** -- automatic macro variable that contains the name of the current session encoding.

**MEXECSIZE** -- system option that specifies the maximum macro size that can be executed in memory.

## Examining the Value of the YEARCUTOFF= System Option with GETOPTION

By Michele Burlew

This example demonstrates how to obtain information about a system option with the SAS function GETOPTION and how to use that information to control the processing of a program.

The GETOPTION function allows you to check SAS options within a DATA step. You can then control processing based on the values of specific options.

The years in the data lines in this example are specified with two-digit years. Assume the code in subsequent steps requires the option YEARCUTOFF to be set to 1920. The DATA step verifies the YEARCUTOFF setting. When the value is not 1920, an ABORT RETURN statement executes and stops the program.

### Original Program

```
data overdue;
  input itemid $ 1-10 +1 checkout mmddyy6. +1
  duedate mmddyy6.;
  format checkout duedate mmddyy10.;
  if _n_=1 then do;
    yc=getoption('YEARCUTOFF');
    if yc ne 1920 then do;
    put '***** Working with dates-YEARCUTOFF not 1920.';
    abort return;
    end;
  end;

  if today() < duedate then output;
datalines;
LIB0386 040301 050301
LIB0387 070500 071900
LIB1064 123000 013001
run;
```

## Using SAS Automatic Variables to Correct Errors

SAS creates SAS automatic variables when a DATA step executes. You can use these variables in your DATA step programming as you would any other variable.

You can write the values of SAS automatic variables to the SAS log to help you analyze your program and data. Your code can test the values of SAS automatic variables to direct conditional execution of DATA step statements.

Automatic variables are temporary and SAS does not store them in the data sets it creates. To save the value of an automatic variable, assign its value to a data set variable.

Table 3.4 lists four automatic variables that are helpful in debugging your DATA steps. When a DATA step executes, SAS always creates the two automatic variables `_ERROR_` and `_N_`. SAS creates the other two variables described in the table when your code includes a BY statement and input from one or more data sets. These four variables are part of the program data vector.

**Table 3.4: SAS Automatic Variables Useful in Debugging DATA Steps**

Automatic Variable	Function
<code>_ERROR_</code>	<p><b>When to use:</b></p> <ul style="list-style-type: none"><li>• Always review the SAS log to look for notes indicating that <code>_ERROR_</code> was set to 1.</li><li>• Set <code>_ERROR_</code> to 1 in your code for errors you want flagged in the SAS log. <code>FIRST.variable</code> Indicates whether the data values being processed are from the first observation in a BY group where: <code>variable</code> is the BY-group variable to examine.</li></ul>
<code>FIRST.variable</code>	<p>Indicates whether the data values being processed are from the first observation in a BY group where:</p> <p><i>variable</i> is the BY-group variable to examine.</p> <p>SAS sets the <code>FIRST.variable</code> to 1 when the current observation is the first observation in the specified BY group. Otherwise, SAS sets this variable to 0.</p> <p><b>When to use:</b></p>

	<ul style="list-style-type: none"> <li>• Include in PUT statements to display information about each BY group.</li> <li>• Use to output specific observations for later review.</li> </ul>
LAST.variable	<p>Indicates whether the data values being processed are from the last observation in a BY group where:</p> <p><i>variable</i> is the BY-group variable to examine.</p> <p>SAS sets the LAST.variable to 1 when the current observation is the last observation in the specified BY group. Otherwise, SAS sets this variable to 0.</p> <p><b>When to use:</b></p> <ul style="list-style-type: none"> <li>• Include in PUT statements to display information about each BY group.</li> <li>• Use to output specific observations for later review</li> </ul>
_N_	<p>Retains the number of iterations of a DATA step.</p> <p>When to use:</p> <ul style="list-style-type: none"> <li>• Include in PUT statements when displaying values of variables in the SAS log.</li> <li>• Use to output specific observations for later review.</li> </ul>

### Working with the `_ERROR_` Automatic Variable

Your code can test the value of the `_ERROR_` automatic variable to detect specific errors and then conditionally execute statements in your DATA steps.

The value of `_ERROR_` is set to 0 at the start of iteration of a DATA step; SAS does not retain the value of `_ERROR_` over iterations. When SAS encounters specific errors, it sets the value of `_ERROR_` to 1. Your code can also set the value of `_ERROR_` to 1.

When either you or SAS sets `_ERROR_` to 1, SAS writes to the SAS log messages and the values in the program data vector.

The value of the system option `ERRORS=` determines the maximum number of observations for which SAS lists complete error messages.

For example, when ERRORS=20, SAS lists in the SAS log error messages for the first 20 observations that have errors. SAS does not identify errors in observations beyond these first 20. SAS writes a message to the SAS log when it reaches the maximum number of errors.

Once you correct the first 20 observations with errors and run the corrected DATA step again, you may find additional observations with errors after those first 20.

When SAS reads data from an external file or from data lines, it prints in the SAS log the column ruler the first time it sets `_ERROR_` to 1.

---

### **About the Author**

Michele Burlew is a self-employed SAS programmer with more than 20 years of experience. She specializes in designing and programming a variety of SAS applications, and she has expertise in using many SAS products. Michele's extensive experience in developing programs and working with end users gives her keen insight into the needs of SAS software users.


Her books are available from the online bookstore.

- [\*Debugging SAS Programs: A Handbook of Tools and Techniques\*](#)
- [\*Reading External Data Files Using SAS: Examples Handbook\*](#)
- [\*SAS Macro Programming Made Easy\*](#)

## Power Talk with Bryan Beverly

*SAS® User for More Than 21 Years*

For 21 years, SAS has been the tool of choice for Bryan Beverly, software architect and team leader at BAE Systems Information Technology. In fact, Beverly admits, “I would not have a career without SAS.”

Beverly’s first introduction to SAS came in 4 at the University of Maryland at Baltimore. Prior to that exposure, he was a devout SPSS user. The addition of SAS came about he says, because “I saw that SAS was flexible, easier to use and the productivity ramp-up time was relatively short.”

Shortly after that initial introduction to SAS, Beverly was asked to develop a series of analytical tables. And as is so typical in these scenarios, the customer needed the data immediately. “I was pulling my hair out trying to develop the tables in SPSS,” explains Beverly. “I had just finished an Intro to SAS course, so I flipped through the manual to see if SAS could produce something close to what the customer wanted. I discovered PROC Tabulate, produced a mock-up, watched the customer’s eyes light up, and the rest is history.”

### **SAS through the years**

During the past 21 years, Beverly began using SAS exclusively. He stopped using SPSS altogether in 1989, primarily, he says, because he found that SAS provided the tool sets and support to solve any type of information delivery problem. “You don’t have to memorize functions, formulas, and the syntax of every procedure because the support for SAS is tremendous. I can use manuals, on-line documentation, call Technical Support, or call on my peers for SAS answers. It’s a great way to work,” comments Beverly.

Beverly has used SAS to support projects in academia, the private sector, as well as the public sector. “This breadth of exposure has been quite helpful in my professional development,” notes Beverly. “The types of products licensed by each company, and the environment in which the work is done, drives the types of applications requested. So, unlike some professions where job migration is a negative, traversing industries is a positive for SAS programmers.”

Earning a bachelor’s degree in sociology at Morgan State University, Beverly’s initial career track was that of a research analyst. But after learning SAS, he evolved into a programmer, systems developer and currently an architect/team leader. After his SAS career began to flourish, Beverly earned a master’s degree in IT management from the Johns Hopkins University and a master’s-level certification in project management from The George Washington University. It’s safe to say that as SAS grew from being a statistical package to an information delivery system, Beverly’s career also grew and developed into what it is today.

## **Growing over time**

“The breadth of the SAS product line I use is often driven by my customers’ requirements,” states Beverly. “During the past seven years, I have used [SAS/AF®](#) with SAS Component Language (SCL), [SAS/IntrNet®](#) and SAS Warehouse Administrator. But Base SAS has given me most of what I needed over the years.”

Beverly believes that the range of products one uses is determined by one’s point of entry into SAS programming. Seasoned programmers have enough code and systems stored away to muscle through most challenges or special programming needs. “Newly minted programmers have access to new tools that do a lot of the tougher work without a lot of extra programming,” he says.

Professional development is an ongoing process for Beverly. He has taken many training courses over the years. Initially, his learning goals were syntax-oriented; now his learning goals are geared toward problem solving. “This is a natural progression,” he comments, “because when you first get started, your audience consists of other programmers. But as you get ‘long in the tooth,’ your audience consists of end-users, sponsors and other stakeholders who influence your funding.”

Beverly likes to accept tough to nearly impossible assignments so that he can expand his SAS knowledge. “Those type of projects force me to find innovative ways of using existing SAS products and tools,” he says. In addition to the tough assignments, he continues to grow with SAS by reading [SAS Press books](#) to discover new functions and procedures. He also keeps up with the SAS product line through the [SAS Web site](#), and he attends [users conferences](#) whenever possible.

## **Passing the knowledge**

Beverly has been attending knowledge-sharing meetings for nearly 15 years, and he does his share of training as well. Over the years, he has served as the coordinator for four in-house users groups, held brown-bag SAS code-sharing sessions, and conducted informal Intro to SAS lessons.

On a broader scale, he presents papers at SAS conferences and publishes technical tips as often as possible in the [SAS Technology Report e-newsletter](#). In fact, he recently submitted a technical tip entitled “[Automated Text Messaging for Cell Phones](#).”

This e-mail facility is currently his favorite tool. “It is nice to have system processes that generate e-mail,” he says. “We expanded this concept a while back, such that our programs send text messages to our cell phones. We are notified if a server fails, disk storage runs out, or if a critical production job fails.”

And it’s nice to know that SAS has played such an important role in Beverly’s career, because if it weren’t for users like him, SAS may not have become the company it is today. We are truly grateful to our long and distinguished list of users.

## **PROC PANEL** (Experimental) System Dependencies

---

The PANEL procedure analyzes a class of linear econometric models that commonly arise when time series and cross-sectional data are combined. The PANEL procedure deals with panel data sets that consist of time series observations on each of several cross-sectional units. The PANEL Procedure is useful where there is clear heterogeneity in the data.

The PANEL procedure provides all of the functionality provided by the TSCSREG Procedure and many new features and improvements. It is available only by download. For documentation for the PANEL procedure, see <http://www.sas.com/statistics/doc.html>.

Note that the PANEL procedure installation requires a local standalone installation of SAS (whether installed through a network install or a local install).

---

### **PROC PANEL DOWNLOAD PACKAGES**

PROC PANEL download package is listed by description in the table below. Selecting the *Request Download* button will allow you to see the license terms and download PROC PANEL.

PROC PANEL (Experimental) for Windows				
Description	Size	Release Date		
<b>PROC PANEL (Experimental)</b>	2.94 MB	2005-09	<a href="#">README</a>	<b>Request Download</b>
<b>PROC PANEL Documentation</b>	1.29 MB	2005-09	<a href="#">README</a>	<b>Request Download</b>
<b>PROC PANEL Sample Program</b>	28 KB	2005-09	<a href="#">README</a>	<b>Request Download</b>

For questions you may have about this **product**, please [contact our Product Support Group](#)

## New Live Web Classes on Data Presentation and Reporting

SAS Education brings you two new Live Web classes to help you present your data in the style or format you want. In the [Creating Detail and Summary Reports](#) course, learn how to use the various reporting tools in SAS to create a specific look-and-feel for your reports. For those who are looking to present their SAS reports in a markup language, consider the course titled [Creating Markup Language Files Using ODS Markup, SAS XLM Libname Engine and the TEMPLATE Procedure](#). Both courses will help you achieve more control over your data presentation and reports.

## Dr. Robert Allison Helps Set New Standards in Dashboard Technology

Driving a car without the dashboard data would be possible but not very practical. Immediate and understandable information about both engine and driver performance is essential to ensuring a safe and problem-free journey. In the business world, computer dashboards are built on the same principle. Company data is provided in display format so that management can quickly gauge various aspects of performance.

### **Current dashboards – well received with some reservations**

Although dashboards have become popular in recent years, most implementations have fallen short. Designers have focused more on “bells and whistles” than on users’ needs for solid information in an easy-to-digest format.

As Stephen Few, Founder and Principal of [Perceptual Edge](#), a consultancy focused on data visualization for analysis and communication notes, “Most dashboards I’ve seen, especially vendor examples, suggest little concern for communication, but a great deal of concern for entertainment.”

Even with the shortcomings, business executives cite dashboards as being critical analytical tools in the new business environment as reported in a recent article in [CRM Today](#). In fact, at the Hyperion Solutions 2005 Conference, 95% of the conference survey respondents noted the importance of dashboards to detect and evaluate data variances.

### **Dashboard challenge**

In the spring, [DM Review](#) issued a [challenge](#) to IT professional by conducting a contest. Coordinated by Few, a columnist for the publication, the contest focus defined “clarity and efficiency” in the composition of digital dashboards as the standard. IT professionals were tasked with developing and designing high quality usable data visualizations with practical workplace applications.

“We ought to judge the efficacy of a dashboard as we would any means of business communication—based on whether the information that people need is presented as clearly and accurately as possible in a way that can be perceived and understood as quickly as possible,” explains Few.

### **SAS steps up to the plate**

SAS management knew that SAS software provides a full range of tools that are perfect for the construction of dashboards. Various SAS products and procedures enable designers to customize dashboards for optimal display of information. SAS enables users to read data directly from spreadsheets (and other sources), re-combine and manipulate data, perform analytics, and then present the data using a variety of interactive graphical techniques.

Management also thought Dr. Robert Allison could do a good job of showcasing the capabilities of SAS. So, SAS Vice President of R&D Paul Kent approached Allison with the competition scenarios.

Allison is competitive by nature and has had a lifelong passion for artistic endeavors. During his childhood he painted lifelike insects on the attic walls in his home, and Allison was early on the technology bandwagon writing his own computer games in high school (1979-1982), long before home computers were common home appliances.

During his undergraduate studies at North Carolina State University, Allison learned the value of layout and visual presentation and began to focus on graphs and maps in graduate school under the direction of Dr. Moon Suh, a professor at North Carolina State University College of Textiles, and Carl Priestland, a chief economist at American Apparel Manufacturers Association of Washington, DC.

### **Building the custom SAS dashboard**

“Dashboards are especially challenging,” notes Allison. “To get the perfect dashboard, you must use very powerful and flexible software. I don't know of any other software I could have used to create this dashboard - SAS allowed me to create this dashboard, end-to-end, using a single SAS program. And another great thing is that I could tweak the SAS program and re-run the entire thing on my desktop PC in less than 15 seconds.”

A variety of SAS procs and products allow designers to customize dashboards for optimal display of information. Allison, for example, used SAS/ACCESS to import the spreadsheet data, PROC DATASETS to rename variables, PROC SQL to combine the datasets and perform calculations, SAS/GRAPH PROC GCHART to draw the bar charts, SAS/GRAPH ANNOTATE for the custom triangle “target” makers, SAS/GRAPH PROC GREPLAY to combine the multiple charts on a single page, and ODS HTML to provide interactive web output with rollover text and drilldowns.

“SAS is the only software I know of that gives you this end-to-end power and the flexibility to visualize your data in any way you want; you are only limited by your imagination,” notes Allison.

### **Allison's SAS dashboard impresses**

When the [DM Review contest](#) “dust” settled, Allison and his custom SAS dashboard ranked tops in the dashboard competition which involved designing a display which would allow “sales executives to know at a glance any areas of sales - both problems and opportunities - that might require their attention.”

Few noted that Allison's dashboard was "not cluttered though rich in information." He found the use of simple bar graphs with consistent quantitative scales helpful for at-a-glance mental digestion of data.

"Robert did an exceptional job of avoiding most of the visual design mistakes that are common in dashboards," writes Few in his analysis of the project.

Allison is honored to be named as the dashboard winner representing SAS and its software and also pleased to contribute to the effort to develop and provide business executives with better display tools for the future.

"Hopefully this contest will both raise users' awareness of the custom graphical capabilities of SAS software, and 'raise the bar' for better dashboards in general," says Allison.

## Data Mining Courses this Fall

### Data rich, Information poor?

It's easy to collect huge volumes of data - social statistics, bank records, biological data, and more - but very hard to pull useful facts and trends out of the heap. Data mining can help spot sales trends, develop smarter marketing campaigns, and accurately predict customer loyalty and predict future outcomes.

---

### Business Knowledge Series courses taught by Industry Experts

- [Data Mining Techniques: Theory and Practice](#) - taught by Michael J. A. Berry or Gordon S. Linoff
- [Credit Scorecard Development and Implementation](#) - taught by Naeem Siddiqi, Business Solution Specialist - Risk, SAS Canada
- [Data Mining Cookbook: Introduction to Effective Predictive Modeling](#) - taught by C. Olivia Parr Rud, Coach, Consultant and author of *Data Mining Cookbook*
- [Predictive Analytics: Getting More Value from Your Data](#) - taught by Jeff Zeanah, Z Solutions, Inc.

### Data mining classes taught by SAS certified instructors using SAS products

- [Data Preparation for Data Mining](#)
- [Predictive Modeling Using Logistic Regression](#)
- [Predictive Modeling Using SAS Enterprise Miner 5.1](#)
- [Advanced Predictive Modeling Using SAS Enterprise Miner 5.1](#)
- [Extending SAS Enterprise Miner 5.1](#)

For the complete list of data mining courses, visit the [curriculum path](#).

## **The Power of PROC FORMAT**

**By:** Jonas V. Bilenas

**List price:** 29.95 USD

124 pages

**ISBN:** 1-59047-573-9

**Publisher:** SAS Press

**Publication Date:** March 2005

### **Description:**

Are you a programmer, statistician, or data analyst tasked with generating reports? Discover how you can put the powerful FORMAT procedure to work for you with The Power of PROC FORMAT. Written in an easy-to-follow tutorial style and illustrated with real-world examples and solutions, this handy guide introduces beginning to intermediate SAS users to the functionality of the FORMAT procedure. Learn how the FORMAT procedure can recategorize data values while doing a variety of tasks, including building user-defined formats and informats, implementing a table lookup in SAS, using the DATA step and other SAS procedures, assigning descriptive labels to data values, creating new variables and finding unexpected values, generating data extracts, and merging data sets.

**SAS Products Addressed:** Base SAS

**Releases:** 9.1.3, 9.1.2, 9.1, 9.0, 8.2

**Operating Systems:** 64-bit Enabled AIX, 64-bit Enabled HP-UX, 64-bit Enabled Solaris, ABI+ for Intel Architecture, AIX, HP-UX, HP-UX IPF, IRIX, Linux, Linux on Itanium, Microsoft Windows for IPF, OS/2, OpenVMS Alpha, OpenVMS VAX, Solaris, Tru64 UNIX, VM/CMS, Windows, Windows NT Workstation, z/OS

[Order today!](#)

## SAS Education partners with University of Memphis

SAS Education provides many training options for students throughout the United States, including live web courses, on-site training and nearly three dozen, state-of-the-art, public training facilities. A recent partnering with the University of Memphis has provided an additional resource for SAS' users. Beginning in November 2005, SAS training will be available on the university's campus giving customers in the surrounding areas easy access to public training. Register for a course at the [Memphis University](#) location and pay before November 1, 2005 to get a free MP3 player!

[Read more](#)

## Leading Geeks


When Paul Glen refers to IT professionals as “geeks,” he isn’t being derogatory. In fact, he considers himself to be one.

And in today’s knowledge-driven, competitive economy, geeks are a key weapon in a business’ arsenal. “As technology continues to drive business productivity and competitiveness, the role of the geek becomes increasingly critical,” Glen states. “Some think that whichever organization attracts and retains the best geeks wins in this environment. They’re only half right.”

After spending years in the IT world in a management capacity, Glen found himself reading a lot of business books. “I became a real consumer of leadership books,” he states. “I always seemed to have a feeling that there was some disconnect between what I was reading and what I was experiencing on the ground.” So to combat this problem, Glen set out to write his own leadership book.

*Leading Geeks: How to Manage and Lead the People Who Deliver Technology*, challenges the conventional wisdom that leadership methods are universal. It gives executives and managers the understanding they need to manage and lead the technologists upon whom they have become so dependent.

As Glen states, much of what managers already know about leadership won’t work with geeks for three primary reasons:

1. Geeks are different from other employees. Most leadership books begin with the fundamental assumption that leadership is a relationship between leaders and followers, and then proceed to focus almost exclusively on the knowledge, attitudes, beliefs, ethics, and behaviors of the leader, as if the nature of followers were irrelevant. But not all followers are alike, and they do not respond to leadership in the same way. Geeks, in particular, are a special group requiring different “care and feeding” from others in an organization.
2. Geekwork, the intricate, hnological-knowledge work that geeks perform, is different from other types of work. Most discussions of leadership assume that leading a group of first-graders on a field trip to a museum is the same as guiding a nation into war. Of course, this simply isn’t true. What managers are trying to lead people to do does, in fact, affect the nature of the relationship between leaders and followers. Glen notes, “Managers need to realize that often in IT work, figuring out what to do can be harder than actually doing it. And knowing when a project is done can be difficult. ‘Done’ is often subjective in the technology world.”
3. Power, the basis of most approaches to leadership, is relatively useless when dealing with geeks. Power is the ability to affect the behavior of others, but geeks don’t deliver value through behavior. They deliver value mostly through thought rather than action, so their behavior has relatively little effect on their productivity. And because most

theories of leadership are based on notions of political, organizational, or social power, they don't work too well with geeks.

### **Changing perceptions**

Glen offers a scenario to get managers thinking about a new and different approach to leadership. A large company is about to launch a new project and the head of IT needs to pull together a particularly well-motivated team. "If you want a team to be really motivated, pick people who are motivated to do the project," says Glen. While this may seem straightforward, managers often create teams based on three factors: who is available, who has the skills and who has done this before?


"We ask all the wrong questions to get a motivated team together," notes Glen. "The first thing you do is ask who wants to work on this type of technology? Who wants to learn a particular role? Who finds the outcome of this project compelling? You look at it through a number of different lenses and decide who will actually be excited to do the work."

Next, Glen suggests leaders make sure their team has a clear understanding of the meaning of the work. Say an IT team has to perform the same tedious work for weeks at a time. It's rather monotonous and not terribly compelling. "But if you tell me the work I am doing will help provide food for 100,000 children, that is meaningful. That's worth all the tedious work," comments Glen.

And finally, in order to pull together a motivated IT team, Glen advises limiting group size. "When you have a team that is working on a complex project, break the team out into subproject teams. That tends to work very well when leading a group," says Glen. "People who are connected to one another are more motivated to work together than people who are part of some large impersonal bureaucracy."

This is just one scenario and one idea in a book chock-full of material on what works with geeks and what doesn't. "Leading geeks can be a difficult job," says Glen. "I hope that by reading my book, managers will come away with the simple idea that leadership in the IT industry is different from leadership elsewhere. And in order to be an effective leader, managers need to adjust."

### **For more information**

To find out more about Glen's book, *Leading Geeks*, visit [www.leadinggeeks.com](http://www.leadinggeeks.com). To hear Glen speak, be sure to sign up for a [free Web seminar](#), "Leading Geeks: What Every IT Organization Should Know about Aligning People, cess, and Technology to Create Strategic Business Value" on Nov. 3 at 2 p.m. ET.

## SAS Enterprise Guide Add-In task for creating a CSF graph

SAS Enterprise Guide includes a graphical user interface for generating code and results for many SAS System procedures. However, because of the complexity involved and the vast number of available procedures, a built-in task for each one is not possible.

To accommodate sites that require functionality not available in built-in tasks, SAS Enterprise Guide offers an interface for incorporating user-built Add-In tasks.

The goal of this sample is to provide the files and steps required for adding an Enterprise Guide Add-In task which can be used to create a Critical Success Factor graph. Note: Microsoft Visual Basic 6 was used to develop this Add-In task.

### CREATE THE ADD-IN TASK FOR SAS ENTERPRISE GUIDE 3.X

Note: These instructions are for release 3.x. The next set of instructions are for release 2.x.

1. Close Enterprise Guide.
2. From your Enterprise Guide client machine, create the path: C:\EGAddIns.
3. Download the attached .zip file and save it to C:\EGAddIns.
4. Extract the .OCX files and .DLL file to C:\EGAddIns.
5. From the Start Menu, select **Programs ➔Command Prompt**.
6. Type CD.. until you are at C:\
7. Type CD EGAddIns.
8. Register *each* .OCX file. For example, enter regsvr32 SASFmts.ocx.
9. Register the CSF.dll file. For example, enter regsvr32 CSF.dll
10. Open Enterprise Guide.
11. From the main menu bar, click **Add-In ➔Add-In Manager**.
12. Click **Browse**. Locate C:\EGAddIns and open CSF.dll.
13. Click **OK** to exit the Add-In Manager.
14. From the main menu bar, click **Add-In ➔CSF**
15. If there is no active data in the project, you will be prompted to select an input data file. After you have selected the data file, the CSF task will appear.  
Note: The data requirements for CSF graphs are very specific. Please refer to the Data Requirements and Sample data sections provided on the Columns tab of the CSF Add-In task.

### CREATE THE ADD-IN TASK FOR SAS ENTERPRISE GUIDE 2.X

Note: These instructions are for release 2.x. The previous instructions are for release 3.x.

1. Close Enterprise Guide.
2. From your Enterprise Guide client machine, create the path: C:\EGAddIns.
3. Download the attached .zip file and save it to C:\EGAddIns.

4. Extract only the Tabctl32.OCX and CSF.DLL files to C:\EGAddIns.
5. From the Start Menu, select **Programs ➔Command Prompt**.
6. Type CD.. until you are at C:\
7. Type CD EGAddIns.
8. Register the Tabctl32.OCX file. Enter regsvr32 Tabctl32.ocx.
9. Register the CSF.dll file. Enter regsvr32 CSF.dll.
10. Open Enterprise Guide.
11. From the main menu bar, click **Tools ➔Customize ➔Add-Ins**.
12. Click **Add**.
13. For **ProgId**, enter csf.graph ➔OK ➔OK ➔OK to exit the Customize window.
14. Select File ➔Exit to close Enterprise Guide.
15. Open Enterprise Guide.
16. From the main menu bar, click **Graph ➔CSF**
17. If there is no active data in the project, you will be prompted to select an input data file. After you have selected the data file, the CSF task will appear.  
Note: The data requirements for CSF graphs are very specific. Please refer to the Data Requirements and Sample data sections provided on the Columns tab of the CSF Add-In task.

## Webcasts and events

### **M2005 Data Mining Conference**

October 24-25

Las Vegas

Join colleagues and the most sought-after experts in data mining for a conference that will change the way your organization uses its most important asset: its data.

### **PNWSUG**

Nov. 6-8

Tacoma, WA

PNWSUG 2005 will be held at the Sheraton Tacoma. Make plans now to network and collaborate with your peers and SAS personnel.

### **National Corporate Counsel Forum on HMDA**

November 15-16

Las Vegas

Prior to the conference, be sure to join SAS on Nov. 14 for a workshop entitled “HMDA Messaging in the Wake of Publication: Creating an Effective Stakeholder Communication Program to Address Exposures.”

### **SAS Offers Free On-Site Workshop**

Gain insight on how to optimize the delivery of your IT-enabled services to your internal and external customers. We'll show you how. (Note: This workshop is not a training class.)