

# Invisible Environmental Awareness Techniques

John K. Troxell, Merck & Co., Inc., Rahway, NJ  
Chang-Min Chen, Merck & Co., Inc., Rahway, NJ

## ABSTRACT

In many SAS programming situations, it pays to be aware of the current state of the SAS session environment. Features added to recent releases of the SAS/BASE product are making this increasingly possible. This paper presents programming techniques that permit programs to obtain the session information necessary to make intelligent choices on the fly, prevent problems, and avoid trampling blindly on the pre-existing state of the session. The techniques discussed here, when used in macros, are "invisible" - they operate behind the scenes and don't leave any traces in the log.

## BACKGROUND

Starting sometime during the SAS 6 releases, and continuing through SAS 8, an increasing number of features have been added to the SAS/BASE product that permit dramatically expanded access to information about the session environment. This improved "environmental awareness" can be useful in many ways. For example, programs can be made to branch according to whether or not a particular dataset, variable, format, macro, or global macro variable exists. Or one can reset options, titles and footnotes to their pre-existing states.

The SASHELP views contain a great deal of useful information about the session environment. Davis (2001) and Thielbar (2002) provide good introductions to the views. The views can be accessed in the usual ways that datasets and views can be. Thielbar illustrates access via PROCS CONTENTS, PRINT, FREQ, and REPORT. Davis presents applications of DATA steps to access the views and PROC SQL to access the dictionary tables that are the counterparts of the views. He recommends the use of the WHERE clause to make the access more efficient.

Techniques discussed by these and other authors leave traces in the log. For example, PROC SQL and DATA\_NULL\_steps leave the usual log entries and notes. This additional complexity in the log may not be desirable. In many cases, macros may need to obtain information about the session environment and act accordingly without bothering the user except when necessary.

## ACHIEVING INVISIBILITY

This paper suggests another approach that leaves no trace in the log: the use of the %sysfunc function to access the views and, where appropriate, the WHERE= and KEEP= operators to make the access more efficient. The %sysfunc function can also access some useful information independently of the SASHELP views. Applications of this technique will be explored in the remainder of the paper.

The following examples were run under SAS 6.12, but the behavior is virtually the same under SAS 8 except for minor differences in the format and content of some log notes.

## DOES A GLOBAL MACRO VARIABLE EXIST?

Before the dictionary tables and SASHELP views were available, certain questions were difficult or impossible to answer. One such problem is to determine whether or not a particular global macro variable exists. Carpenter (1998) discusses a SAS macro %symchk, posted at the SAS web site, that solves this problem.

We have modified the %symchk macro to print a message about whether or not the global macro variable is found. This modified macro is called %symchk1. We compare %symchk1 to two other macros: %symchk2, based on the dictionary tables along the lines suggested by Davis; and %symchk3, based on our suggested approach. In each approach, we will look at what "extra stuff" gets printed to the SAS log in two cases: when the global variable exists, and when it does not.

### Approach 1 (the original approach):

```
%macro symchk1(name=);
%local name;
%let name=%upcase(&name);
%if %nrquote(&&&name)=%nrstr(&)&name %then
    %put symchk1: global macro variable
    &name does not exist;
%else %put symchk1: global macro variable
    &name exists;
%mend symchk1;
```

When global macro variable &tmp exists %symchk1(name=tmp) results in:

```
symchk1: global macro variable TMP exists
```

When global macro variable &tmp does not exist, %symchk1(name=tmp) results in:

```
WARNING: Apparent symbolic reference TMP not resolved.
symchk1: global macro variable TMP does not exist
```

Only one extra line (bolded here) gets put to the log, in the event that &tmp doesn't exist. Unfortunately, it is a **WARNING**: message of a type that usually indicates a bug. This might be alarming.

### Approach 2 (the PROC SQL approach):

```
%macro symchk2(name=);
%local name check;
%let name=%upcase(&name);
%let check=;
proc sql noprint;
    select distinct name
    into :check
    from dictionary.macros
    where scope='GLOBAL' and name="&name";
quit;
%if %length(&check) gt 0 %then
    %put symchk2: global macro variable
    &name exists;
%else %put symchk2: global macro variable
    &name does not exist;
%mend symchk2;
```

When &tmp exists, %symchk2(name=tmp) results in:

```
MPRINT(SYMCHK2): PROC SQL NOPRINT;
MPRINT(SYMCHK2): SELECT DISTINCT NAME INTO
:CHECK FROM DICTIONARY.MACROS WHERE
SCOPE='GLOBAL'
AND NAME="TMP";
MPRINT(SYMCHK2): QUIT;
NOTE: The PROCEDURE SQL used 0.02 seconds.
symchk2: global macro variable TMP exists
```

When &tmp does not exist, %symchk2(name=tmp) results in:

```
MPRINT(SYMCHK2): PROC SQL NOPRINT;
MPRINT(SYMCHK2): SELECT DISTINCT NAME INTO
:CHECK FROM DICTIONARY.MACROS WHERE
SCOPE='GLOBAL'
AND NAME="TMP";
NOTE: No rows were selected.
MPRINT(SYMCHK2): QUIT;
NOTE: The PROCEDURE SQL used 0.04 seconds.
symchk2: global macro variable TMP does not
exist
```

If option MPRINT is set, the PROC SQL code is echoed back to the log. Also, regardless of the MPRINT setting, we get a note about the time used by the macro. More importantly, if &tmp does not exist, we always get a message **NOTE: No rows were selected**. If this technique were embedded into a macro for the purpose of establishing whether or not &tmp existed, the presence of this note in the log might not be desirable. Of course we could set options NOMPRINT and NONOTES, but then we would have an options statement in the log, which also might raise eyebrows.

### Approach 3 (the invisible approach):

```
%macro symchk3(name=);
%local name dsid rc rc2;
%let name=%upcase(&name);
%let dsid =
%sysfunc(open(sashelp.vmacro(where=(scope='GLOBAL'
and name="&name") keep=scope name)));
%let rc = %sysfunc(fetch(&dsid));
%let rc2 = %sysfunc(close(&dsid));
%if &rc ne -1 %then
%put symchk3: global macro variable
&name exists;
%else %put symchk3: global macro variable
&name does not exist;
%mend symchk3;
```

When global macro variable &tmp exists %symchk3(name=tmp) results in:

```
symchk3: global macro variable TMP exists
```

When global macro variable &tmp does not exist, %symchk3(name=tmp) results in:

```
symchk3: global macro variable TMP does not
exist
```

Nothing goes to the log except what we want. This technique is therefore "invisible" and particularly well suited to embedding within macros when the need is to ascertain a fact without side effects in the SAS log.

## DOES A MACRO EXIST?

Troxell (2002) discusses an example of the use of PROC SQL to determine whether or not a particular macro has been defined in the current session. Here is a simplified version of that example:

```
%let macnames=;
proc sql noprint;
select distinct objname
into :macnames separated by ' '
from dictionary.catalogs
where objtype eq 'MACRO'
and objname eq 'LEVENE';
quit;
%if "&macnames" eq "LEVENE" %then %put Macro
LEVENE found;
%else %put Macro
LEVENE not found;
```

If macro %levene has been defined during the session, then this code results in the following in the log:

```
MPRINT(CHECKIT): PROC SQL NOPRINT;
MPRINT(CHECKIT): SELECT DISTINCT OBJNAME INTO
:MACNAMES SEPARATED BY ' ' FROM
DICTIONARY.CATALOGS WHERE OBJTYPE EQ 'MACRO'
AND OBJNAME EQ 'LEVENE';
MPRINT(CHECKIT): QUIT;
NOTE: The PROCEDURE SQL used 0.55 seconds.
Macro LEVENE found
```

The code was embedded in a macro called %checkit. Option MPRINT was set, so we see the PROC SQL code. We are also informed that PROCEDURE SQL used 0.55 seconds. Finally we get the message that macro LEVENE was found.

If macro %levene has not been defined during the session, then the following appears:

```
MPRINT(CHECKIT): PROC SQL NOPRINT;
MPRINT(CHECKIT): SELECT DISTINCT OBJNAME INTO
:MACNAMES SEPARATED BY ' ' FROM
DICTIONARY.CATALOGS WHERE OBJTYPE EQ 'MACRO'
AND OBJNAME EQ 'LEVENE';
NOTE: No rows were selected.
MPRINT(CHECKIT): QUIT;
NOTE: The PROCEDURE SQL used 0.61 seconds.
Macro LEVENE not found
```

Again we get the procedure code echoed because of the MPRINT setting, and the additional notes that are not a function of MPRINT: the note about time used to run PROC SQL and the note about no rows being selected. We also get the expected note about macro Levene not being found.

Now let's look at an "invisible" approach:

```
%let found = -1;
%let dsid =
%sysfunc(open(sashelp.vcatalog(where=(objtype eq
'MACRO' and objname eq 'LEVENE') keep=objtype
objname)));
%let found = %sysfunc(fetch(&dsid));
%let rc=%sysfunc(close(&dsid));
%if &found ne -1 %then %put Macro LEVENE found;
%else %put Macro LEVENE not
found;
```

If macro %levene has been defined during the session, then this code results in the following in the log:

```
Macro LEVENE found
```

If macro %levene has not been defined during the session, then the following appears:

```
Macro LEVENE not found
```

Nothing goes to the log except the intended message. This is true no matter what the MPRINT setting is. By using the "invisible" approach, we exercise control over what goes into the log and in what circumstances.

## DETERMINE ALL WORK DATASET NAMES THAT START WITH A SPECIFIED STRING

In our final example, imagine that we want to know the names of all temporary WORK library datasets whose names start with a given string, say "abc."

One very visible way to do this would be to execute PROC DATASETS or PROC CONTENTS, and obtain the list by searching the resulting log, listing, or dataset output.

Another visible approach would be to use PROC SQL as follows:

```
%local dsnames;
proc sql noprint;
    select memname into :dsnames separated by
    ,
    from dictionary.tables
    where libname eq 'WORK'
    and memtype eq 'DATA'
    and memname like 'ABC%';
quit;
%put dsnames=&dsnames;
```

As an aside, the trailing % in the constant ABC%, in conjunction with the like comparison operator, matches all dataset names that start with ABC (case insensitive). In many actual situations ABC would be parameterized to generalize the macro to be able to search for any starting string, not just ABC. In such a case, if the starting string value abc is contained in a variable called &root, then one could do this:

```
%let root = %upcase(&root)%str(%);
```

and, on the affected line,

```
and memname like "&root"
```

which would resolve to :

```
and memname like "ABC%"
```

In any event, the above code results in the usual log messages. To be merciful to the reader, we set option NOMPRINT this time. If datasets abc, abc1, abc2, abcdefg, abc\_text, abc\_old, and junk exist, then the above code results in:

```
NOTE: The PROCEDURE SQL used 0.0 seconds.
dsnames=ABC ABC1 ABC2 ABCDEFG ABC_TEXT ABC_OLD
```

&dsnames could then be tokenized into as many local macro variables as necessary. PROC SQL could instead put each dataset name found into its own individual macro variable. The problem with that approach is that it is hard to know how many such variables to declare local before invoking PROC SQL. PROC SQL creates global macro variables unless one has set them local before they are created.

In the event that there are no matching datasets, the code results in:

```
NOTE: No rows were selected.
NOTE: The PROCEDURE SQL used 0.02 seconds.
dsnames=
```

We see once again that the PROC SQL approach is not invisible.

Now here is the invisible approach:

```
%local nds dsid rc;
%let nds = 0;
%let dsid =
%sysfunc(open(sashelp.vstable(where=(libname='W
ORK' and memname='ABC'))));
%do %while (%sysfunc(fetch(&dsid)) eq 0);
    %let nds = %eval(&nds+1);
    %local ds&nds;
    %let ds&nds = %sysfunc(getvarc(&dsid,2));
%end;
%let rc=%sysfunc(close(&dsid));
%put number of datasets that exist: &nds;
%if &nds gt 0 %then %do i=1 %to &nds;
    %put ds&i = &&ds&i;
%end;
```

Comparison operator "=" is used to match any dataset names starting with string 'ABC'. The =: comparison operator can't be used in the WHERE clause in PROC SQL but as we have seen the equivalent can be accomplished using like and %.

We don't use KEEP= here because the sashelp.vstable view has only two columns and we need them both. The second argument of the getvarc function is hardcoded to 2 because we know that memname is the second column in the view.

When the same datasets as in the PROC SQL example exist, we get the following in the log:

```
number of datasets that exist: 6
ds1 = ABC
ds2 = ABC1
ds3 = ABC2
ds4 = ABCDEFG
ds5 = ABC_TEXT
ds6 = ABC_OLD
```

In the event that there are no matching datasets, the code results in:

```
number of datasets that exist: 0
```

With the "invisible" approach, the desired information is obtained without anything appearing in the log except what is desired.

## CONCLUSION

There are several possible approaches to obtaining information about the SAS session environment. We have presented here a general solution that allows programs to obtain information without distracting the user with unnecessary log notes.

## REFERENCES

Carpenter, Art. 1998. "Carpenter's Complete Guide to the SAS Macro Language". Cary, NC: SAS Institute, Inc., 1998. 242 pp.

Davis, Michael. 2001. "You Could Look It Up: An Introduction to the SASHELP Dictionary Views". Proceedings of SUGI 26.

Thielbar, Melinda. 2002. "The SASHELP Library: It Really Does Help You Manage Data".  
<http://www.sas.com/service/techtips/bitsandbytes/sashelp.html>

Troxell, John K. 2002. "Bulletproofing and Knowledge Encapsulation in Statistical Macros". Proceedings of PharmaSUG 2002.

## **CONTACT INFORMATION**

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

John K. Troxell  
Merck & Co., Inc.  
RY34-A320  
P.O. Box 2000  
Rahway, NJ 07065-0900  
Work Phone: (732) 594-0475  
Fax: (732) 594-6075  
Email: [john\\_troxell@merck.com](mailto:john_troxell@merck.com)  
Web: n/a

Chang-Min Chen  
Merck & Co., Inc.  
RY34-A320  
P.O. Box 2000  
Rahway, NJ 07065-0900  
Work Phone: (732) 594-1992  
Fax: (732) 594-6075  
Email: [changmin\\_chen@merck.com](mailto:changmin_chen@merck.com)  
Web: n/a