Paper 339-2013

A Macro to Verify a Macro Exists

Rick Langston, SAS Institute Inc., Cary, NC

ABSTRACT

Although the %SYSMACEXIST function can do macro existence checking, it is limited to pre-compiled macros. This paper describes the %macro_exists macro, which will verify that a specified macro will be found if invoked. The macro will search for pre-compiled macros and will search all autocall libraries to verify the existence.

INTRODUCTION

A SAS user had discussed with me the need to determine that a particular macro (we will call it %ABC) exists before attempting to invoke %ABC. After all, if %ABC does not exist, errors will result. Out of that discussion came the macro named %macro_exists, which is described herein.

HOW MACROS ARE FOUND

Before describing the macros, we should first understand what the macro facility does when %ABC is seen in SAS code. It first checks to see if the member ABC exists in work.sasmacr, which is the temporary macro catalog. If the member is found there, then that is assumed to be the ABC macro to use. If not, the macro facility checks to see if the SASMSTORE option has been given. This option specifies a libref, and the sasmacr catalog associated with that libref is examined to determine if the ABC member is found. (Note that the MSTORED option must also be in effect for this to happen). If the ABC member is still not found, the macro facility checks sashelp.sasmacr as well. If ABC is not found in any of these catalogs, then the macro facility checks for the SASAUTOS option. This option is a list of filerefs (unquoted) and pathnames (quoted) enclosed with parentheses. Each fileref and pathname is checked, in order of specification, for the file abc.sas. If that file is found, then it is effectively %INCLUDEd and the %ABC macro is expected to be found therein.

BUILDING BLOCK MACROS

We will need to replicate this process within our own implementation of %macro_exists. Before describing that macro, however, we should first examine two macros that are the building blocks for %macro_exists. These macros are %member_exists and %compiled_macro_exists.

First, we have %member_exists. Here is the code for this macro:

```
/* Given a fileref and a memname and memtype, we attempt to open the
   member of the directory (catalog or file system directory). We
   set &member_found to 1 if it can be opened, 0 if not. */
%macro member_exists(fileref,memname,memtype);
%global member_found;
%let member_found = 0;
data _null_;

     *-----open the directory and proceed if it can be opened-----*;
     handle = dopen("&fileref.");
     if handle ne 0;

     *-----open the member and set the macro variable based on result-----*;
     mem_handle = mopen(handle,"&memname..&memtype.",'i');
     call symputx('member_found',mem_handle ne 0);

     *-----close the member if it were opened successfully-----*;
     if mem_handle then rc = fclose(mem_handle);

     *-----close the directory-----*;
     rc = dclose(handle);
     run;
%mend member_exists;
```

The macro is given a fileref, which refers to a directory-based entity. This can be an actual directory on UNIX or Windows, or it can be via an access method such as FTP or CATALOG. The DOPEN function has to succeed, which can happen only with a proper directory-based fileref. If DOPEN succeeds, then we try to open the specified member and member type for input via the MOPEN function. The &member_found macro variable is set to 1 if the member can be opened, or 0 if it cannot. We are not interested in reading anything from the member, just ensuring that we can open it successfully. We simply close it via FCLOSE once we open it. The DCLOSE function is also called, to ensure proper closure after the DOPEN call.

Our other building block macro is %compiled_macro_exists. Here is the code for this macro:

```
/* Given a macro name, we determine if it has already been
   compiled. We first look in work.sasmacr, then in the sasmacr
   referenced by sasmstore (if given) and then in sashelp.sasmacr. */
%macro compiled_macro_exists(macro_name);
%global member_found;

*-----try work.sasmacr first to see if the compiled macro is there-----*;
filename maclib catalog "work.sasmacr";
%member_exists(maclib,&macro_name.,macro);
filename maclib clear;
%if &member_found %then %goto done;

*-----try sasmacr referenced by sasmstore if it were specified-----*;
%let sasmstore_option = %sysfunc(getoption(sasmstore));
%if %sysfunc(getoption(mstored))=MSTORED and %length(&sasmstore_option) > 0 %then
%do;
filename maclib catalog "&sasmstore_option..sasmacr";
%member_exists(maclib,&macro_name.,macro);
%end;
%if &member_found %then %goto done;

*-----try sashelp.sasmacr last-----*;
filename maclib catalog "sashelp.sasmacr";
%member_exists(maclib,&macro_name.,macro);

%done: ;
%mend compiled_macro_exists;
```

This macro makes use of the %member_exists macro, and provides the logic to emulate what the macro facility does in searching for a pre-compiled macro in the various catalogs. It uses %member_exists to find the member in work.sasmacr by using a fileref associated with the CATALOG access method. If not found there, it uses %member_exists to find the member in the SASMSTORE= option location. If not found there, it uses %member_exists to find the member in sashelp.sasmacr. The &member_found macro variable is set to 1 or 0 based on whether the member was found.

THE %MACRO_EXISTS MACRO DESCRIBED

Now that we have seen the definitions of %compiled_macro_exists and %member_exists, we can see the definition for %macro_exists, which fully emulates the macro facility for locating a macro definition.

```
/* Given a macro name, we determine if it would be executed if
   invoked. It may already be compiled or would be brought in
   via autocall. We first check to see if it's compiled. If not
   in compiled form, then we try to open the .sas file using
   the list of all sasautos directories. If it is found, we
   %include the file so the macro definition should take place,
   then we go looking for it as a compiled macro again. The global
   macro variable member_found is set to 1 if found and 0 if not. */

%macro macro_exists(macro_name);
%global member_found;

*-----see if the macro already exists in compiled form-----*;
%compiled_macro_exists(&macro_name.);
```

```
%if &member_found %then %goto done;

/* If the macro does not exist in compiled form, we need to search for
   it in each of the autocall directories until it is found. The sasautos
   option contains a parenthesized list of pathnames (quoted) and filerefs
   (not quoted). For a pathname, we have to make a fileref for it. We use the
   %member_exists macro for the fileref and the macro name with the .sas
   extension. Note that this is all generated as part of a macro definition
   so that we can %goto around the rest of the code once a member is found. */

filename &sascode_fileref. temp;
data _null_; file &sascode_fileref.;

     /*-----macro definition-----*/
     put '%macro ' "&process_sasautos_name." ';';
     put '%global member_found;';

     /*-----get SASAUTOS option and blank out the parentheses-----*/
     text=getoption('sasautos');
     if substr(text,1,1)='(' then substr(text,1,1)=' ';
     l=length(text);
     if substr(text,l,1)=')' then substr(text,l,1)=' ';

     /*-----loop through all the tokens in SASAUTOS-----*/
     i=0;
     do while(1);
        i+1;

        /*-----read a quoted token (pathname) or non-quoted token (fileref)-----*/
        x=scan(text,i,' "''','q');
        if x=' ' then leave;

        /*-----change double-quoted to single-quoted to avoid macro substitution—*/
        if substr(x,1,1)='"' then do;
           substr(x,length(x),1)=' ';
           substr(x,1,1)=' ';
           x=tranwrd(x,'""','"');
           x=tranwrd(x,"'","''");
           x="'"||substr(x,2,length(x)-1)||"'";
           end;

        /*-----use FILENAME statement to create fileref if needed-----*/
        length fileref $8;
        if x=:"'" then do;
           fileref="&myautos_fileref.";
           put 'filename ' "&myautos_fileref. " x ';';
           do_clear=1;
           end;
        else do;
           fileref=x;
           do_clear=0;
           end;

        /*-----issue macro code to handle the fileref-----*/
        put '%member_exists(' fileref ',' "&macro_name." ',sas);';
        put '%if &member_found %then %do;';
        put '%include ' fileref "(&macro_name.)" '/source2; run;';
        if do_clear then put 'filename ' fileref ' clear;';
        put '%compiled_macro_exists' "(&macro_name.)" ';';
        put '%goto done_looking;';
        put '%end;';
        end;
    /*-----complete the macro definition-----*/
    put '%done_looking:;';
    put '%mend ' "&process_sasautos_name" ';';
    run;

/*-----include the generated code to define the macro-----*/
%include &sascode_fileref./source2; run;
filename &sascode_fileref. clear;

/*-----now invoke the macro-----*/
%&process_sasautos_name.;

%done: ;
%mend macro_exists;
```

First, we see if the macro already exists in compiled form. The %compiled_macro_exists macro does that for us. If it doesn't exist in compiled form, we will need to search the SASAUTOS list. This is done by executing a DATA step to extract the file specifications. (This proved to be simpler than attempting to do this all within macro code). Note that the DATA step is actually generating SAS code and emitting it to a file and %INCLUDEing this file. The SAS code is actually a macro definition itself, named per the user-provided &process_sasautos_name macro variable. Note the use of the SCAN function with the 'q' argument, which allows for quoted strings to be provided intact even if blanks exist within the quotation marks. Once the quoted string is obtained, the code ensures that the quoted string uses single quotation marks instead of double quotation marks to avoid any problems with macro variables or macro specifications being expanded. (For example, a pathname of "A&B" might cause problems, but 'A&B' would not). If the SASAUTOS token is not quoted, then it is a fileref; otherwise, we create a FILENAME statement using the quoted path. We then invoke the %member_exists macro for that fileref, looking for a .sas file of the proper name. If the file is found, then it is %INCLUDEd, as it would be via autocall processing. Then, %compiled_macro_exists is tried for the macro to ensure that it was compiled as part of the %INCLUDE process.

GENERATED CODE FROM %MACRO_EXISTS

For example, if we have invoked %macro_exists with the macro name of abc, and if the SASAUTOS option has the value (A 'xyz'), which means a fileref A followed by a pathname 'xyz', the generated macro—using the name myname—will be:

```
%macro myname;
%global member_found;
%member_exists(A,abc,sas);
%if &member_found %then %do;
%include A(abc)/source2; run;
%compiled_macro_exists(abc);
%goto done_looking;
%end;
filename myfile 'xyz';
%member_exists(myfile,abc,sas);
%if &member_found %then %do;
%include myfile(abc)/source2; run;
filename myfile clear;
%compiled_macro_exists(abc);
%goto done_looking;
%end;
%done_looking:;
%mend myname;
```

This macro code first looks for the abc.sas member in the A fileref. If it is found, the member is %INCLUDEd and then we confirm that the %abc macro was actually compiled as part of that %INCLUDE. (If %abc was not defined, the macro facility would not continue looking in other SASAUTOS entries). If abc.sas were not found in A, then we associate the fileref MYFILE with the 'xyz' path, then look for abc.sas in that location. As with A, we %INCLUDE abc.sas if it is found and check for %abc having been defined.

EXAMPLE OF THE USE OF %MACRO_EXISTS

Here is an example of using the %macro_exists macro. We will look for the non-existent macro called blabblah. The &member_found macro variable should be set to 0 because it is not found. We then define the macro blahblah and then try the %macro_exists macro again, this time expecting to see &member_found set to 1.

```
%macro_exists(blahblah);
%put does blahblah exist? &member_found;

%macro blahblah; %put this is blahblah; %mend;

%macro_exists(blahblah);
%put does blahblah exist? &member_found;
```

ACKNOWLEDGMENTS

I would like to thank David H. Abbott of the Veterans Administration for suggesting that such a feature be provided in the SAS System. He addressed the issue in his SESUG 2012 presentation. I have provided this macro to him and he has found it useful for his processing needs.

REFERENCES

Abbott, David. 2012. “Make SAS Macros Safe for Other to Use”. SESUG 2012, Durham, North Carolina. analytics.ncsu.edu/sesug/2012/BB-08.pdf

CONTACT INFORMATION

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

Rick Langston
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Rick.Langston@sas.com