PharmaSUG2010 - Paper TU01

OBJECT_EXIST:

A Macro to Check if a Specified Object Exists

Jim Johnson, Independent Consultant, North Wales, PA

ABSTRACT

This paper describes a macro designed to quickly tell the program whether or not an object exists. The types of objects the macro can detect are SAS® datasets, external files, open libraries, open filerefs, macros, macro variables, formats, informats, and specific variables in a dataset.

The macro is designed to either print a message to the log and set a user error flag, or set a user defined 'found flag' and allow the calling program to handle the results of the OBJECT_EXIST macro.

OBJECT_EXIST is a useful macro for applications, utilities, and one-time only programs because all the search algorithms are in one place, validated, and being a macro, only the section of code needed to search for a specific item will be included in the job stream.

INTRODUCTION

"There are 100 different ways to do everything in SAS." This paper describes only one way, which by no means suggests that it is the only way. The reader should feel free to explore other techniques that SAS offers to do the same things shown in this paper.

The OBJECT_EXIST program exists and has been validated; however the code is proprietary and cannot be shared in its entirety. The techniques and code shown herein offer the reader a glimpse into the overall program and process. The reader is encouraged to use the information here to build their own version of the program.

The programs and techniques shown in this paper are relative to SAS version 9.1.3.

Dictionary Tables
Using Proc SQL and the dictionary tables is far more efficient than using the data step and the sashelp views. Accessing tables is always more efficient than accessing views. Only Proc SQL can address the tables directly through the *dictionary* libname.

> *When you query a DICTIONARY table, SAS gathers information that is pertinent to that table. Depending on the DICTIONARY table that is being queried, this process can include searching libraries, opening tables, and executing SAS views. Unlike other SAS procedures and the DATA step, PROC SQL can improve this process by optimizing the query before the select process is launched. Therefore, although it is possible to access DICTIONARY table information with SAS procedures or the DATA step by using the SASHELP views, it is often more efficient to use PROC SQL instead.* – **SAS 9.1.3 Language Reference: Concepts - Dictionary Tables and Performance (Online Documentation)**

Therefore example code in this paper will use Proc SQL and the DICTIONARY libname to access the dictionary tables.

There are a number of good papers on the SAS dictionary tables and several have been referenced in the RECOMMENDED READING section of this paper.

PARAMETERS

The macro needs three parameters:

a) The name of the object. This is required since the name of the object you seek must be known.

b) The type of object you seek. This is required so the program knows where to look for your object.
c) An optional macro variable flag name to populate indicating whether or not the object was found. If provided, a macro variable of the name specified will be created and populated with a 0 or 1 to indicate if the object was found. If this parameter is left blank, when an object is not found the program will write an error message to the log. Use of the flag allows your program to handle the circumstances around the presence or absence of your object.

The types of objects that can be identified includes:

a) SAS datasets.
b) SAS dataset variables.
c) Directory paths.
d) Specific files within a directory.
e) File references (fileref) assigned through a SAS FILENAME statement.
f) Library references (libref) assigned through a SAS LIBNAME statement.
g) SAS formats.
h) SAS informats.
i) SAS macros.
j) SAS macro variables.

The output of the macro depends on the use of the found optional flag parameter.

1) If the found flag parameter **is not** used,
    a. the output will be nothing in the event the object exists, or
    b. will be an error message printed to the log if the object does not exist.
2) If the found flag parameter **is** used,
    a. the flag will be set to 1 if the object is found, or
    b. the flag will be set to 0 if the object is not found. No messages will be printed to the log if the found flag parameter is used.

DATA SOURCES & TECHNIQUES

Most of the objects can be located using dataset functions. Those that cannot conveniently be located using dataset functions can be found using the dictionary tables.

There are a number of resources for more information on dataset functions and dictionary tables, and details of each will not be discussed in this paper. The RECOMMENDED READING section at the end of this paper list some valuable papers that explain these resources.

SAS DATASETS

The `exist` dataset function can be used to locate SAS datasets

```
%sysfunc(exist(&object_name,data))
```

when 0, the dataset does not exist, when 1, the dataset does exist. The object name can be a one- or two-level name.

```
%sysfunc(exist(one,data))
%sysfunc(exist(work.one,data))
%sysfunc(exist(input.one,data))
```

Optionally, it may be of some value to know whether or not the dataset contains any observations. This can be accessed through dataset functions

```
%let dsid = %sysfunc(open(&object_name));
%let nobs = %sysfunc(attrn(&dsid,nobs));
%let rc   = %sysfunc(close(&dsid));
%if &nobs = 0 %then %do;
   ...
```

SAS DATASET VARIABLES

The SAS dataset variable name can be specified as a two- or three-level name. A three-level name would be specified as `<libname>.<dataset name>.<variable name>`. A two-level name would be specified as `<dataset name>.<variable name>` and the libname portion will be assumed to be `'work'`. A one level variable name should be rejected by the macro since there is not enough information to locate the variable.

The object name needs to be parsed into each level so the macro can look for each component, specifically that the libname exists, then the dataset exists, and then that the variable exists. If any of the levels specified do not exist, the macro will print a message to the log or set the user specified found flag.

```
%let level3 = %upcase(%scan(&object_name,-1,.));
%let level2 = %upcase(%scan(&object_name,-2,.));
%let level1 = %upcase(%scan(&object_name,-3,.));
%if %nrbquote(&level1) = %then %let level1 = WORK;
```

The use of the negative sign with the `%scan` function causes the function to operate from the right side of the object name instead of the left side. For instance, if the object name was specified as

```
input.demog.race
```

then the `level3` macro variable will be obtained by selecting the right most value delimited by a period, in this case the `level3` macro variable will be set to `RACE` (after the `%upcase` function).

The use of `%nrbquote(&level1)` technique is the safest method for identifying a missing macro variable in that it properly responds when the value is blank, %str(), contains macro triggers (& or %), or is a very long character string. It fails only when the value is a very long number. The RECOMMENDED READING section of this paper lists an excellent paper that details the research around this and other techniques.

The `exist` dataset function can be used to locate the libname and dataset combination parsed from the object name

```
%sysfunc(exist(&level1..&level2,data))
```

A result of 0 means the dataset does not exist, a 1 means the dataset exists.

Once the libname and dataset combination have been verified to exist, the specific variable can be located with other dataset functions

```
%let dsid = %sysfunc(open(&level1..&level2,i));
%let n    = %sysfunc(attrn(&dsid,nvar));
```

The code above opens the dataset and determines the number of variables in the dataset. Then a loop can go through the variables to see if one is the variable you seek

```
%do i = 1 %to &n;
   %if %sysfunc(varname(&dsid,&i))) = &level3 %then
   ...
```

For a more detailed look at the variable, the variable type and length can be obtained once the variable is found

```
%let v_type   = %sysfunc(vartype(&dsid,&i));
%let v_length = %sysfunc(varlen(&dsid,&i));
```

With this information, the macro can optionally compare the found variable attributes to expected attributes. For instance, if you were expecting a character variable of length 30, you can confirm that is the kind of variable you found. This paper leaves it to the reader to develop the appropriate parameter input and logic.

DIRECTORY PATHS

A directory differs from a file in that a file is considered an object within a directory.

The `filename` dataset function can be used to locate a directory

```
%let fileref = dummy;
%let rc = %sysfunc(filename(fileref,"&object_name"));
```

This code will attempt to open the object. The object name needs to be quoted. For some reason these statements cannot be combined as shown below without reporting a false positive from the `dopen` statement to follow.

```
%let rc = %sysfunc(filename(dummy,"&object_name")); /* DOES NOT WORK */
```

The return code from the two statement code above should be a zero, which means the file reference was assigned. This return code provides no feedback on the existence of the directory because the file reference will be assigned whether or not the specified directory exists, so further steps are necessary. Open the directory using a dataset function

```
%let did = %sysfunc(dopen(&fileref));
```

If `&did` > 0 the object exists. If `&did` = 0 the object does not exist.

In all cases, it is wise to close the objects you have opened rather than leave the objects hanging around to cause potential problems elsewhere.

```
%let rc = %sysfunc(dclose(&did));
```

FILES

A file differs from a directory in that a file is considered an object within a directory.

The `fileexist` dataset function can be used to locate a file but will not tell the whole story because the `fileexist` dataset function will return a positive response to both a directory and an object within the directory.

```
%sysfunc(fileexist(&object_name))
```

The function above will return a 0 if the object does not exist and a 1 if the object does exist. A response of 0 can be accepted as is, but a response of 1 will require additional logic because if the named object is erroneously given as a directory as opposed to a file, `fileexist` will still return a 1 indicating the object exists. The additional logic that must be applied will distinguish between a file and a directory by checking for a directory.

```
%let fileref = dummy;
%let rc = %sysfunc(filename(fileref,"&object_name"));
%let did = %sysfunc(dopen(&fileref));
%let rc = %sysfunc(dclose(&did));
```

The final question for the program then becomes

```
%if %sysfunc(fileexist(&object_name)) = 0  OR
   (%sysfunc(fileexist(&object_name)) = 1 and &did > 0)
```

If the first condition is true, then the object simply does not exist. If the second condition is true then the object exists as a directory, not a file.

FILE REFERENCES

A file reference, or fileref, refers to the reference assigned through a SAS FILENAME statement.

The `fileref` dataset function can be used to locate an open file reference.

```
%sysfunc(fileref(&object_name))
```

If the return code = 0 then the file reference exists.
If the return code > 0 then the file reference does not exist.

If the return code < 0 then the file reference exists, but the pathname is in question. This can happen when a filename statement is provided a non-existent pathname or the physical path has been removed, the file reference will exist, but it will not actually point to anything.

SAS FORMATS

SAS Formats can be located using the dictionary table *formats*. Formats can be temporary and stored in the WORK directory or permanent and stored in libraries. SAS also provides a number of built-in formats. The FMTSEARCH option is used to specify where SAS should look for permanent format libraries. The setting of the FMTSEARCH option is available through the dictionary table *options*.

```
proc sql;
   select setting into :fmtsearch
   from   dictionary.options
   where  optname = 'FMTSEARCH';
```

The list provided in this macro variable need to be formatted so that they can be plugged into a Proc SQL WHERE statement IN clause. The list generated above can contain some special characters such as double quotes and parenthesis. They need to be removed to strip the list down to just the catalog names, and then double quote each name.

```
select '"' || upcase(tranwrd(compress(trim(setting),'"()'),' ','" "')) || '"' into
:fmtsearch
```

The statement above will convert a value such as

```
(WORK LIBRARY)
```

into a value like

```
"WORK" "LIBRARY"
```

The list selected from this query above can be fed into a WHERE statement IN clause

```
proc sql;
   select objname
   from   dictionary.formats
   where  libname in (&fmtsearch " ") /* blank gets the SAS provided formats */
   and    fmtname = compress("&object_name",'.')
   and    fmttype = 'F';
```

Adding a blank to the libname list will recover the SAS provided formats such as `left`, `substr`, and `upcase`.

The `fmttype = 'F'` limits the selection to formats, and excludes informats. If the first character of the `objname` is a dollar sign ($) then the format is a character format, otherwise it is a numeric format.

SAS INFORMATS

SAS Informats can be located in exactly the same way as formats except

```
and    fmttype = 'I'
```

Similarly, if the first character of the objname is a dollar sign ($) then the informat is a character informat, otherwise it is a numeric informat.

LIBRARY REFERENCES

A library reference, or libref, refers to the reference assigned through a SAS LIBNAME statement.

The libref dataset function can be used to locate an open library reference.

```
%sysfunc(libref(&object_name))
```

If the return code = 0 then the library reference exists.
If the return code > 0 then the library reference does not exist.
If the return code < 0 then the library reference exists, but the pathname is in question. This can happen when a LIBNAME statement is provided a non-existent pathname or the physical path has been removed, the library reference will exist, but it will not actually point to anything.

MACROS

Macros are stored as catalogs and can be located using the dictionary table *catalgs*. Macros can be temporary objects that your program compiled as it was executed and are stored in the WORK directory. Other macros can be stored compiled macros and accessible through the SASAUTOS option. Therefore the OBJECT_EXIST macro should find out what libraries have been specified in the SASAUTOS option. This is attainable through the dictionary table *options*.

```
proc sql;
   select setting into :sasautos
   from   dictionary.options
   where  optname = 'SASAUTOS';
```

The list provided in this macro variable need to be formatted so that they can be plugged into a Proc SQL WHERE statement IN clause. The list generated above can contain some special characters such as double quotes and parenthesis. They need to be removed to strip the list down to just the library name, and then double quote each name.

```
select '"' || upcase(tranwrd(compress(trim(setting),'"()'),' ','" "')) || '"' into
:sasautos
```

The statement above will convert a value such as

```
(CORPLIB PROJLIB SASAUTOS)
```

into a value like

```
"CORPLIB" "PROJLIB" "SASAUTOS"
```

The list selected from this query can be fed into a WHERE statement IN clause

```
proc sql;
   select objname
   from   dictionary.catalgs
   where  libname in (&sasautos "WORK")
   and    objname = "&object_name"
   and    objtype = "MACRO";
```

The libname list has "WORK" hard coded so that any temporary macros generated by a recently run program will also be searched.

MACRO VARIABLES

Macro variables can be located in the local and global symbol tables which are accessible through dataset functions

```
%sysfunc(symglobl(&object_name))
%sysfunc(symlocal(&object_name))
```

The symglobl and symlocal functions will return a 1 if the macro variable exists and a 0 if it does not exist.

The macro variable name should be provided in the OBJECT_NAME parameter **without** the leading ampersand (&),

otherwise the macro variable will be resolved and the OBJECT_EXIST macro will process the resolved value. For example, if the macro variable `population` was set to

```
Intent to Treat
```

and the call to OBJECT_EXIST included the ampersand

```
%object_exist(object_name = &population
              ,object_type = macrovar);
```

then the macro variable would resolve before reaching the dataset function and would produce a statement like this

```
%sysfunc(symglobl(Intent to Treat))
```

which will clearly provide unexpected results.

CONCLUSION

Any program, be it an application, a utility, ore even a one time only program that has to look for an object can benefit from this macro. The use of the 'found flag' allows the calling program to control how to handle the presence or absence of the object. Certainly it is easy enough to simply put the necessary logic in your code to locate the item you desire, but what if you look for more than one object? Or what if you have to look for different types of objects? The advantage of a validated macro that locates objects is clear. There may be a little more overhead making calls to the macro, but the benefit of trusting an existing validated macro is worth the trade-off.

RECOMMENDED READING

- Chang Chung and John King (2009), **"Is This Macro Parameter Blank?"**, *Proceeding of the 2009 SAS Global Forum*, Cary, NC: SAS Institute, Inc.
- Yue Ye and Yong Lin (May 2000), **"Using SAS Functions in Data Steps"**, *Proceedings of the 2000 Conference of the Pharmaceutical Industry SAS Users Group,* Cary, NC: SAS Institute, Inc.
- Thornton, Patrick (2008), **"SAS DICTIONARY: Step by Step"**, Proceedings of the 2008 Western Users of SAS Software Conference.
- Lafler, Kirk Paul (2009), **"Exploring DICTIONARY Tables and SASHELP Views"**, *Proceedings of the 2009 Conference of the Pharmaceutical Industry SAS Users Group,* Cary, NC: SAS Institute, Inc.
- DiIorio, Frank, **"Summary of SAS Dictionary Tables and Views"**, www.codecraftersinc.com, http://www.codecraftersinc.com/pdf/DictionaryTablesRefCard.pdf

ABOUT THE AUTHOR

Jim Johnson has been programming with SAS in the Pharmaceutical Industry since 1986. He has presented at many local, regional, and national conferences and has been teaching in the SAS Certificate Program at Philadelphia University since its inception in 1997. Jim has a reputation as a "problem solver" and efficiency enthusiast. His recent work includes large SAS systems, writing programs that write programs, an SDTM compliance verification system, infrastructure programming, and advanced validation and documentation skills.

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

Jim Johnson
jimmy2960@verizon.net