Paper 5115-2020

Work Smarter, Not Harder: Learning to Live without Your X

Amit Patel; Lewis Mitchell

ABSTRACT

Within the SAS® community, a great deal of the population are experts in their own fields, but those aren't always technical roles. After supporting hundreds of users over a number of years, we realized there is one question that always comes up — how do we live without our X? With X-commands, pretty much anything is possible, and that unlimited potential makes both administrators and auditors nervous. As environments evolve and controls tighten, the loss of X-commands is all but inevitable for most teams; but never fear, there are alternatives! This paper will discuss how to replace the following: getting file listings; making and deleting directories and files; changing file permissions; Microsoft Windows compression. By harnessing the power of built-in SAS functions as well as other functionality, we are able to replicate much of what users have lost out on when faced with X-commands being disabled, enabling users to finally move on from their X.

INTRODUCTION

There are many benefits which come with having X-commands enabled, but due to tighter controls around operating models, some companies are restricting access to X-commands. The SAS language has a variety of tools which can support users in everyday activities, but sometimes these may not be perceived to be as easy to use as X-commands. In this paper we discuss how the macros provided have been developed, to make all commands easily accessible to all users. Wherever possible the macros should act as a suitable replacement for X-commands.

All macro code (including additional functional macros which are not discussed in this paper) can be found in the appendix.

1. UNIX DIRECTORY AND FILE OPERATIONS

The below macros have been designed to support certain file operations which are unavailable/less robust/difficult to perform without using X-commands. We will explain some of the base functions used within the code, what to expect for outputs and also discuss optional parameters which may be useful.

Not included, although existing in all macros, is the option parameter *put=Y/put=N.* By default this parameter is a Y which provides a slightly more verbose output. Where this parameter is passed as an N, limited output is provided.

Restricted - External

1.1 %SYS_DELETE

The sys_delete macro is used to delete any single file or empty directory. One path parameter is passed to the macro, with the return code and return message passed back to SAS.

Syntax:
`%sys_delete(path=/c/production/data/a/b);`

Output 1 shows a failure message within the log.

```
INFO: The macro return code is 20047
ERROR: /c/production/data/a/b is not empty and cannot be deleted.
```

Output 1. Failure log output from sys_delete

Output 2 shows a success message within the log.

```
INFO: The macro return code is 0
```

Output 2. Success log output from sys_delete

The function and corresponding return values used in this macro can be called in a dataset in the following manner:

1) Assign a filename to the file or directory to delete

   `filename _in_ "%superq(path)";`

2) Within a data step, call the function *fdelete* to delete the file or empty directory and capture the return code within the variable *macro_rc* and the return message within the variable *macro_msg*.

   `macro_rc = fdelete('_in_');`

   `macro_msg = sysmsg();`

1.2 %SYS_DIR_CREATE

The sys_dir_create macro is used to create recursive directories for any given path. One path parameter is passed to the macro to create the missing directories along the directory structure.

Syntax:
`%sys_dir_create(path=/c/production/data/a/b/c/d/e)`

Output 3 shows a successful messages within the log.

```
Directory already exists: /c
Directory already exists: /c/production
Directory already exists: /c/production/data
Directory does not exist. Creating now: /c/production/data/a
Directory does not exist. Creating now: /c/production/data/a/b
Directory does not exist. Creating now: /c/production/data/a/b/c
Directory does not exist. Creating now: /c/production/data/a/b/c/d
Directory does not exist. Creating now: /c/production/data/a/b/c/d/e
```

Output 3. Log output from sys_dir_create

Restricted - External

The function used to create these directories is *dcreate*. Within a data step this can be called using the line:

```
new_path = dcreate(dir, parent_path);
```

1.3 %SYS_MOVE

The sys_move macro is used to move any single file or directory and requires two parameters to pass. The parameter *path* is used to specify the full source filename with the parameter *newpath* being used to specify the full target filename.

Syntax:

```
%sys_move(path=/c/production/data, newpath=/c/production/data2);
```

On success the macro return code will be 0.

The function used to perform the move is *rename*. Within a data step this can be called using the line:

```
macro_rc = rename("/c/production/data", "/c/production/data2", 'file');
```

1.4 %SYS_FILE_COPY

The sys_file_copy macro is used to copy any single file and requires two parameters to pass. The parameter *path* is used to specify the full source filename with the parameter *newpath* being used to specify the full target filename. The advantage with this macro is that any file extension can be copied and not just SAS file extensions.

Syntax:

```
%sys_file_copy(path=/c/production/economic.xls,
newpath=/c/production/economic_data.xls, put=Y);
```

On success the macro return code will be 0.

To perform the copy the input file is read in a byte at a time in binary mode and written out to the new location.

Assign the input and output filenames:

```
filename _in_ "/c/production/economic.xls";
filename _out_ "/c/production/economic_data.xls";
```

Open the input and output files in input/output modes respectively:

```
filein = fopen('_in_','I',1,'B');
fileout = fopen('_out_','O',1,'B');
```

Loop through the file and write out each byte one at a time:

```
rec = '20'x;
do while(fread(filein)=0);
```

Restricted - External

```
  macro_rc = fget(filein,rec,1);
  macro_rc = fput(fileout,rec);
  macro_rc = fwrite(fileout);
end;
```

Close the input and output files:

```
macro_rc = fclose(filein);
macro_rc = fclose(fileout);
```

1.5 %SYS_LS

The sys_ls macro produces an output as close as possible to a Unix file listing command, usually run as *ls –l*. The one required parameter is *ls_path* which refers to the directory being listed.

Syntax:
```
%sys_ls(ls_path=/c/production/data);
```

By default the output will be given in dataset format. This will provide all pieces of data available for each directory and file directly within the directory being queried. To do this the source directory needs to be opened as an object and then the individual objects targeted to extract the information in the following steps:

1) Assign a file reference to the primary directory noted as the variable FileSpecification:

   ```
   rc1 = filename(pref, FileSpecification);
   ```

2) If the file reference is assigned without error then open the directory using the function *dopen*. Assign a directory identifier value *pid* when opening the directory:

   ```
   pid = dopen(pref);
   ```

3) Get the parent directory information using the *pid*

4) Count the number of child members within the parent directory using

   ```
   dnum = dnum(pid);
   ```

5) Loop through each object child object and assign *MemberName* as the object name.
   a. If the object is a directory then we will need to follow steps 2 and 3 above to retrieve the information. This will then have a new directory identifier and the information can be queried using the function *dinfo*
   b. If the object is a file then the parent *pid* can be used to open this specific file and assign a file identifier value labelled *fid* to it:

      ```
      fid = mopen(pid, MemberName, 'I');
      ```

      The function *finfo* can retrieve the necessary information.

Restricted - External

Key columns are shown below and how they are derived. Three columns are derived from the location of the file. These are:

a) FileSpecification: Full filename which is parent directory plus member name

b) ParentDirectory: Parent directory passed within ls_path parameter

c) ShortFilename: Member name

Other information is gained by querying the member object directly using the *finfo* function against the specific file identifier value assigned to the member object.

d) OwnerName = finfo(fid,'Owner Name');

e) GroupName = finfo(fid,'Group Name');

f) AccessPermission = finfo(fid,'Access Permission');

   Within the output this is translated into a column which shows octal permissions

g) FileSize = input(finfo(fid,'File Size (bytes)'),32.);

h) LastModified = finfo(fid,'Last Modified');

   This is returned in a combined date and time format. Within the macro these are split out into fields named *LastModifiedDate* and *LastModifiedTime*.

Lastly there are fields which specify extra pieces of information:

i) Type: This is set to 'f' for files and 'd' for directories

j) Readable: This is set to 'Y' if the child object can be read or else 'N'

k) Sysmsg: Any value in this field would indicate an error returned on querying the child member

When querying directories in a similar manner the *finfo* function should be replaced with *dinfo* and *fid* value should be replaced with that of the open directory. File size is not able to be retrieved for directories.

Other optional parameters on the macro are:

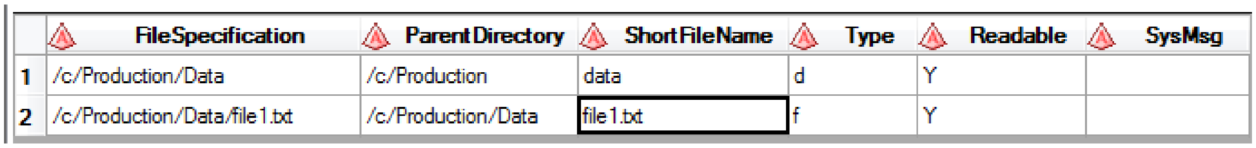
a) Format=

   By default this is assign a value of L to return all information. If this is set to any other value, then all information listed in labels d to h above are not returned.

b) Out_ref=

   By default the output dataset is named work._ls_output_ but this can be overridden with any user defined value

%sys_ls output where output format is shortened. i.e. format ^= 'L'

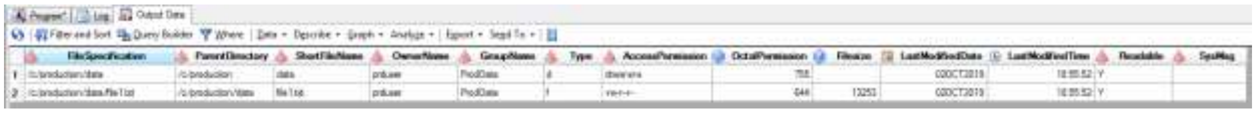


	FileSpecification	ParentDirectory	ShortFileName	Type	Readable	SysMsg
1	/c/Production/Data	/c/Production	data	d	Y	
2	/c/Production/Data/file1.txt	/c/Production/Data	file1.txt	f	Y	

Figure 1. Limited output for sys_ls

Restricted - External

%sys_ls output where output format is full. i.e. format = 'L'



	FileSpecification	ParentDirectory	ShortFileName	OwnerName	GroupName	Type	AccessPermission	OctalPermission	Filesize	LastModifiedDate	LastModifiedTime	Readable	SysMsg
1	/c/production/data	/c/production	data	prduser	ProdData	d	drwxr-xr-x	755		03OCT2019	18:55:52	Y	
2	/c/production/data/file1.txt	/c/production/data	file1.txt	prduser	ProdData	f	-rw-r--r--	644	13253	03OCT2019	18:55:52	Y	

Figure 2. Full output for sys_ls

1.6 %SYS_FIND

The sys_find macro produces an output as close as possible to a Unix find command which produces a file listing output, usually run as *find . -ls*. The difference between this and a Unix *ls -l* command is the ability to traverse directories multiple levels deep. The one required parameter is *find_path* which refers to the directory being listed.

Syntax:

```
%sys_find(find_path=/c/production/data);
```

This macro uses all the sys_ls macro logic as a base and builds on this by storing any directories found whilst querying and adding them to a list to then query further. The additional parameter which can be passed in this macro is *max_depth=*. By default this is limitless and all items under a particular directory can be found. You can limit the number of directories deep the query should go by using this option. The field *depth* will show how many levels down each file was located in reference to the source directory passed within the *find_path* parameter.

By default the *out_ref* parameter outputs the final dataset to work._find_output_.

2) CHECK GROUPS AND PERMISSIONS

2.1 %SYS_CHECK_PERMISSIONS

The security model within an environment can, on occasions, be complex to fit the requirements of the business model being applied. With no easy method on how to replicate access of a fellow user or query permissions on directories for access, the macro provided will allow a user with existing access to retrieve the permissions and Unix groups needed for read or write access to given areas.

Running this macro requires a single parameter which will be the directory which needs to be accessed.

Syntax:

```
%sys_check_permissions(path=/c/production/data/mortgages/model);
```

Assuming access is already granted to the user running the code, the output will be shown in output 4 and will show the individual read and write groups needed:

```
-----------------------------------| ---------| ----------| ------------
FILESPECIFICATION:                 | OWNER:   | GROUP:    | PERMISSIONS:
-----------------------------------| ---------| ----------| ------------
/c                                 | admuser  | SASData   | drwxr-x---
/c/production                      | prduser  | ProdData  | drwxr-x---
/c/production/data/                | prduser  | ProdData  | drwxr-xr-x
/c/production/data/mortgages       | prduser  | ProdData  | drwxr-xr-x
/c/production/data/mortgages/model | prduser  | ModelData | drwxr-x---
-----------------------------------| ---------| ----------| ------------

INFO: To Read from "/c/production/data/mortgages/model" you need the
following groups:
SASData
ProdData
ModelData

INFO: Unable to WRITE to "/c/production/data/mortgages/model"
INFO: Please contact the owner "prduser" regarding permissions "drwxr-x---"
```

Output 4. Log output from sys_check_permissions

The primary step in retrieving the permissions is using the macro %sys_ls as explained under file utilities earlier in this document. This will loop through each directory within the directory structure within the *path* parameter passed. This appends the results together for further analysis.

The parameters kept by the %sys_ls call are:

FileSpecification – Full directory path
ShortFileName – Name of the directory being queried this loop
OwnerName – Unix owner name
GroupName – Unix group name
AccessPermission – Unix permissions

With minor modifications, it is possible to remove the %sys_ls call and replace this with a table with all possible directories within an environment, ensuring it contains all the necessary columns and sets a variable *type = 'd'*. It would then allow all users to query directory permissions regardless of access.

Restricted - External

3. ZIPPING FILES IN SAS

There are a few methods of zipping files such as gz, bzip and bzip2 which are available on Unix systems, however without X-commands these are hard to access. A common format used on Windows and Unix is the .zip extension and SAS has in-built functionality to enable files to be zipped, viewed and unzipped through the filename method. These have been built into the macros below as well as several layers of checks/outputs to supports users when zipping.

For example purposes, our working directory will be defined as &_path which will resolve to the location /c/production/data.

3.1 %SYS_ZIP_MEMBER_ADD

In order to create a new zip file or add files to an existing zip file the same method can apply. Using the sys_zip_member_add macro it is possible to pass across a series of parameters and write a file into a zip archive.

Syntax:
```
%sys_zip_member_add(fs=&_path/testfile.txt, zip_fs=&_path/testzip.zip,
zip_mem=testmember.txt, overwrite=Y, delete=N);
```

In the above example we have passed 5 parameters into the macro:

a) fs: This is the FileSpecification (or full filename) of the file to add into the zip

b) zip_fs: This is the FileSpecification (or full filename) of the zip file which either already exists or you would like to be created

c) zip_mem: This will be the new name of the file once it is in the zip file. It does not have to be the same as the original filename

d) overwrite: This determines on whether you would like to overwrite an existing zip member of the same name. You can pass Y/N

e) delete: This determines on whether you would like to delete the original file you are moving into the zip file. You can pass Y/N

Log output when running example above.

```
INFO: fs        = /c/production/data/testfile.txt
INFO: zip_fs    = /c/production/data/testzip.zip
INFO: zip_mem   = testmember.txt
INFO: overwrite = Y
INFO: delete    = N
CHECK: The file does exist and the zip parent directory does exist.
CHECK: The overwrite option is set to Y which requires no further checks.

OUTCOME: Member has been added to zip file.
```

Output 5. Log output from sys_zip_member_add

The primary steps in adding files to a zip file are shown below. These involve assigning a filename to the original file to be added, and then assigning a filename utilizing the *zip* type and specifying the member where the file should be written to:

Restricted - External

```
filename fs "&_path/testfile.txt";
filename zip_mem zip "&_path/testzip.zip " member="testmember.txt";


data _null_;
  infile fs recfm=n;
  file zip_mem recfm=n;
  input byte $char1. @;
  put byte $char1. @;
run;
```

3.2 %SYS_ZIP_MEMBER_CONTENTS

When needing to view the members of a zip file it is possible to get the listing and output the names to 1 of 3 locations.

Syntax:
```
%sys_zip_contents(zip_fs=&_path/testzip.zip, type=DATA,
out_ref=work._zip_contents_output_);
```

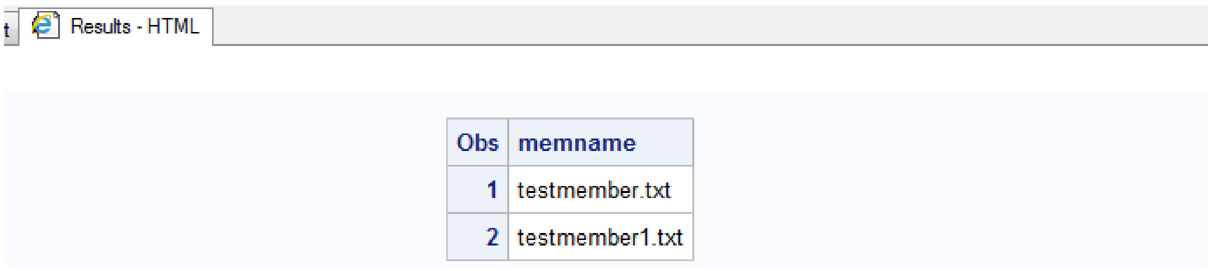
In the above example we have passed 3 parameters into the macro:

a) zip_fs: This is the FileSpecification (or full filename) of the zip file which is being queried

b) type: Determines the output method which can be DATA, LOG or OUTPUT

c) out_ref: Determines the name of the dataset where the results can be written do

Output when type=OUTPUT:



Obs	memname
1	testmember.txt
2	testmember1.txt

Figure 3. HTML output of SYS_ZIP_MEMBER_CONTENTS when type=OUTPUT

Output when type=LOG

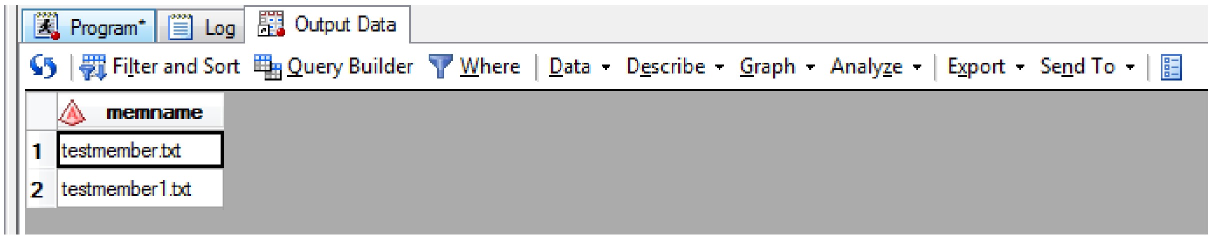
```
INFO: fs        = /c/production/data/testfile.txt
INFO: zip_fs    = /c/production/data/testzip.zip
INFO: zip_mem   = testmember.txt
INFO: overwrite = Y
INFO: delete    = N
CHECK: The file does exist and the zip parent directory does exist.
CHECK: The overwrite option is set to Y which requires no further checks.

OUTCOME: Member has been added to zip file.
```

Output 6. LOG output of SYS_ZIP_MEMBER_CONTENTS when type=LOG

Restricted - External

Output when type=DATA



	memname
1	testmember.txt
2	testmember1.txt

Figure 4. Dataset view of output of SYS_ZIP_MEMBER_CONTENTS when type=DATA

The primary steps in viewing the zip members are shown below. These involve assigning a filename to the zip file and opening the file using the function *dopen*. By using *dnum(fid)* to count the number of objects and then *dread* to extract the member name we can then output this into its required form.

```
filename zip_mem zip "&_path/testzip.zip";
data work._zip_contents_ (keep=memname);
  length memname $512;
  fid=dopen("zip_fs");
  if fid=0 then
    stop;
  memcount=dnum(fid);
  do i=1 to memcount;
    memname=dread(fid,i);
    output;
  end;
  rc=dclose(fid);
run;
```

3.3 %SYS_ZIP_MEMBER_DELETE

On occasions it may be necessary to delete individual members from a zip file.

Syntax:
`%sys_zip_member_delete(zip_fs=&_path/testzip.zip, zip_mem=testmember1.txt);`

In the above example we have passed 2 parameters into the macro:

a) zip_fs: This is the FileSpecification (or full filename) of the zip file to delete from

b) zip_mem: This is the member to delete

Output note within log when successful:

```
INFO: zip_fs    = /c/production/data/testzip.zip
INFO: zip_mem   = testmember1.txt
CHECK: The zip file exists.
CHECK: The zip member exists.
DELETE: Successfully deleted zip member testmember1.txt.
```

Output 7. LOG output of SYS_ZIP_MEMBER_DELETE

Restricted - External

To delete a zip member a filename should be assigned to that particular member and then the data step function *fdelete* used to remove the member.

```
filename zip_mem ZIP "/c/production/data/testzip.zip" member="
testmember1.txt";

data _null_;
  rc = fdelete('zip_mem');
run;
```

3.4 %SYS_ZIP_MEMBER_EXTRACT

When extracting an existing file from within a zip file we can do so by running the following:

Syntax:
```
%sys_zip_member_extract(fs=&_path/testextract.txt, zip_fs=&_path/testzip.zip,
zip_mem=testmember.txt, overwrite=N, delete=N);
```

In the above example we have passed 5 parameters into the macro:

a) fs: This is the FileSpecification (or full filename) of the file once it has been extracted from the zip file.

b) zip_fs: This is the FileSpecification (or full filename) of the zip file which is to be extracted from.

c) zip_mem: This is the name of he zip member to be extracted.

d) overwrite: This determines whether an existing file on the drive should be overwritten if it has the same name of the file being extracted. Y/N can be passed.

e) delete: This determines on whether to delete the zip member once it has been extracted. Y/N can be passed.

Log output when running example above.

```
NFO: fs        = /c/production/data/testextract.txt
INFO: zip_fs    = /c/production/data/testzip.zip
INFO: zip_mem   = testmember.txt
INFO: overwrite = N
INFO: delete    = N
CHECK: The output file directory does exist and the zip file does exist.
CHECK: The zip member does exist.
CHECK: The overwrite option is set to N which requires further checks.
CHECK: The output file does not already exist.

OUTCOME: Member has been successfully extracted to output file.
DELETE: Delete flag is preventing deletion of testmember.txt.
```

Output 8. Log output from sys_zip_member_extract

The primary steps involved in extracting files from a zip file to an accessible drive are shown below. These involve assigning a filename to the file on the drive and also a filename utilizing the type *zip* and specifying the member which is to be extracted. The source file can then be read from and written to the target.

Restricted - External

```
filename fs "&_path/testextract.txt";
filename zip_mem zip "&_path/testzip.zip" member="testmember.txt";

data _null_;
   infile zip_mem lrecl=256 recfm=F length=length eof=eof unbuf;
   file fs lrecl=256 recfm=N;

   input;
   put _infile_ $varying256. length;

   return;
   eof:
   stop;
run;;
```

CONCLUSION

As we can see from the examples provided, there are many substitutes for commands which may not seem straight forward at first. By providing these as ready-made solutions, it can help in many areas of development work, ensuring both users and admins alike are able to run the same code. It has also become apparent that although simple SAS solutions aren't available for every task, there are ways to ensure users can still run certain OS commands in the form of compiled C functions. While compiled functions are out of scope for this paper, as access is limited to SAS administrator management, solutions are available and all isn't lost without your X!

CONTACT INFORMATION

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

Amit Patel
Amit.xc.patel@gmail.com

Lewis Mitchell
l.j.mitchell-06@hotmail.com

Restricted - External

APPENDIX A – SUPPORTING MACROS USED IN CODE

```
%macro sys_log_print(in_ref=, in_keep=_all_) /
des = 'Print a dataset to the log in a readable format (max 256 characters).'
;

  %*** Remember the current value of options prior to changing them. ;
  %options_remember(options=linesize mlogic mprint notes pagesize symbolgen,
options_id=sys_log_print, put=N);

  options linesize=256 nomlogic nomprint nonotes pagesize=1024 nosymbolgen;

  %*** Define local macro variable(s). ;
  %local y z;

  %if %sysfunc(exist(&in_ref.)) = 1 %then %do;

    %*** Get the contents for the specified columns ;
    proc contents noprint
      data = &in_ref. (keep=&in_keep.)
      out = work._sys_log_print_contents_
    ;run;

    %*** Order by varnum. ;
    proc sort
      data = work._sys_log_print_contents_;
      by varnum;
    run;

    %*** Count the number of columns. ;
    proc sql noprint;
      select count(*)
      into :sys_log_count
      from work._sys_log_print_contents_
    ;quit;

    %let sys_log_count = %sysfunc(strip(&sys_log_count.));

    %if &sys_log_count. > 0 %then %do;

      %*** Read the name of the variables into a macro variable array. ;
      proc sql noprint;
        select
          name,
          type
        into
          :sys_log_name_1-:sys_log_name_&sys_log_count.,
          :sys_log_type_1-:sys_log_type_&sys_log_count.
        from work._sys_log_print_contents_
      ;quit;

      %*** Determine the max utilised length for each variable. ;
      proc sql;
      create table work._sys_log_max_ as
        select
          %do y=1 %to &sys_log_count.;
            %if &y. > 1 %then %do;
```

```
              ,
            %end;
            %if &&sys_log_type_&y.. = 1 %then %do;
              max(lengthn(compress(put(&&sys_log_name_&y..,32.)))) as
sys_log_len&y.
            %end;
            %else %if &&sys_log_type_&y.. = 2 %then %do;
              max(lengthn(&&sys_log_name_&y..)) as sys_log_len&y.
            %end;
          %end;
        from &in_ref. (keep=&in_keep.)
      ;quit;

      %*** Determine the max length comparing the max utilised length and the
header. ;
      proc sql noprint;
        select
          %do y=1 %to &sys_log_count.;
            %if &y. > 1 %then %do;
              ,
            %end;

max(sys_log_len&y.,length(cats(upcase("&&sys_log_name_&y.."),':'))) as
sys_log_len&y.
          %end;
        into
          %do y=1 %to &sys_log_count.;
            %if &y. > 1 %then %do;
              ,
            %end;
            :sys_log_len&y.
          %end;
        from work._sys_log_max_
      ;quit;

      %*** Strip out spaces in the length macro variables. ;
      %do y=1 %to &sys_log_count.;
        %let sys_log_len&y. = %sysfunc(strip(&&sys_log_len&y..));
      %end;

      %*** Calculate the overall line length for all output. ;
      %let sys_log_row=0;
      %do y=1 %to &sys_log_count.;
        %let sys_log_row = %eval(&sys_log_row. + &&sys_log_len&y.. + 3);
      %end;
      %let sys_log_row = %sysfunc(strip(&sys_log_row.));

      %*** Generate a string of dashes to match each variable. ;
      %do y=1 %to &sys_log_count.;
        %let sys_log_dash&y.=;
        %do z=1 %to &&sys_log_len&y..;
          %let sys_log_dash&y.=&&sys_log_dash&y..-;
        %end;
      %end;

      %*** Create a dataset of dashes. ;
      data work._sys_log_dashes_ (keep=_line_);
        length
          _line_ $&sys_log_row.
          %do y=1 %to &sys_log_count.;
            &&sys_log_name_&y.. $&&sys_log_len&y..
          %end;
          ;
        %do y=1 %to &sys_log_count.;
          &&sys_log_name_&y.. = "&&sys_log_dash&y..";
        %end;
        _line_ = cat(
          %do y=1 %to &sys_log_count.;
            %if &y. > 1 %then %do;
              ," | ",
            %end;
            &&sys_log_name_&y..
          %end;
        );
      run;

      %*** Create a dataset of headers. ;
      data work._sys_log_header_ (keep=_line_);
        length
          _line_ $&sys_log_row.
          %do y=1 %to &sys_log_count.;
            &&sys_log_name_&y.. $&&sys_log_len&y..
          %end;
          ;
        %do y=1 %to &sys_log_count.;
          &&sys_log_name_&y.. = upcase("&&sys_log_name_&y..:");
        %end;
        _line_ = cat(
          %do y=1 %to &sys_log_count.;
            %if &y. > 1 %then %do;
              ," | ",
            %end;
            &&sys_log_name_&y..
          %end;
        );
      run;

      %*** Create a dataset of data. ;
      data work._sys_log_data_ /*(keep=_line_)*/;
        length
          _line_ $&sys_log_row.
          %do y=1 %to &sys_log_count.;
            sys_temp_&y. $&&sys_log_len&y..
          %end;
          ;
        set &in_ref. (keep=&in_keep.);
        %do y=1 %to &sys_log_count.;
          %if &&sys_log_type_&y.. = 1 %then %do;
            sys_temp_&y. = compress(put(&&sys_log_name_&y..,32.));
          %end;
          %else %if &&sys_log_type_&y.. = 2 %then %do;
            sys_temp_&y. = &&sys_log_name_&y..;
          %end;
        %end;
        _line_ = cat(
          %do y=1 %to &sys_log_count.;
```

```
          %if &y. > 1 %then %do;
            ," | ",
          %end;
          sys_temp_&y.
        %end;
      );
    run;

    %put ;

    %*** Print a row of dashes to the log. ;
    data _null_;
      set work._sys_log_dashes_;
      put _line_;
    run;

    %*** Print a row for the header to the log. ;
    data _null_;
      set work._sys_log_header_;
      put _line_;
    run;

    %*** Print a row of dashes to the log. ;
    data _null_;
      set work._sys_log_dashes_;
      put _line_;
    run;

    %*** Print the data to the log. ;
    data _null_;
      set work._sys_log_data_;
      put _line_;
    run;

    %*** Print a row of dashes to the log. ;
    data _null_;
      set work._sys_log_dashes_;
      put _line_;
    run;

    %put ;

  %end;

  %*** Delete the temp table(s). ;
  proc datasets lib=work nolist;
    delete
      _sys_log_print_contents_
      _sys_log_max_
      _sys_log_dashes_
      _sys_log_header_
      _sys_log_data_
      ;
    run;
  quit;

%end;
```

```
  %options_reset(options=linesize mlogic mprint notes pagesize symbolgen,
options_id=sys_log_print, put=N);

%mend sys_log_print;


%macro options_remember(options=, options_id=, put=Y) /
des = 'Record the current value for any options passed.';

  %*** Set macro variables to be local to this macro to avoid conflicts. ;
  %local _i_ _options_ num_options;

  %*** Clean the input of any spurious spaces. ;
  %let _options_=%sysfunc(strip(%sysfunc(compbl(&options.))));

  %*** Count how many options have been passed. ;
  %let num_options=%sysfunc(countw(&_options_.));

  %*** If the put flag is used, then put information to the log. ;
  %if %sysfunc(upcase(&put.)) = Y %then %do;
    %put INFO: There have been %sysfunc(strip(&num_options.)) options
passed.;
    %put INFO: The options that will be remembered are "&_options_.".;
  %end;

  %*** Loop for each option. ;
  %do _i_=1 %to &num_options.;

    %*** Set looped macro variables to be local to this macro to avoid
conflicts. ;
    %local option&_i_. string&_i_. result&_i_.;

    %*** Find the nth option in the string passed. ;
    %let option&_i_.=%sysfunc(scan(&_options_.,&_i_.));

    %*** String will be populated with the option name wrapped with
underscores. ;
    %let string&_i_.=%sysfunc(compress(_&&option&_i_.._));

    %*** For each option, record the current value of the option. ;
    %let result&_i_.=%sysfunc(getoption(&&option&_i_..));

    %*** If the value of the option is numeric then prefix the value with
option name and = ;
    %if
%sysfunc(compress(%sysfunc(substr(%sysfunc(strip(&&result&_i_.)),1,1)),,kd))
^= %then %do;
      %let result&_i_.=&&option&_i_.=&&result&_i_.;
    %end;

    %*** If the macro variable does not already exist then do...;
    %if %symexist(&&string&_i_..) = 0 %then %do;

      %*** Use string to generate a global macro variable... ;
      %global &&string&_i_..;

      %*** ...that will contain the current value of the option ;
      %let &&string&_i_..=&&result&_i_..;
```

Restricted - External

```
      %*** Test whether the options id macro variable is populated. ;
      %if %sysevalf(%superq(options_id)=,boolean) = 0 %then %do;

        %*** Set additional looped macro variables to be local to this macro 
to avoid conflicts. ;
        %local string_id&_i_.;

        %*** String id will be populated with the option name with "_id" as a 
suffix... ;
        %*** ...and the entire string wrapped with underscores. ;
        %let string_id&_i_.=%sysfunc(compress(_&&option&_i_.._id_));

        %*** Use string id to generate a global macro variable... ;
        %global &&string_id&_i_..;

        %*** ...that will contain the value of the options id ;
        %let &&string_id&_i_..=%superq(options_id);

      %end;

      %*** If the put flag is used, then put information to the log. ;
      %if %sysfunc(upcase(&put.)) = Y %then %do;
        %put INFO: Option "&&option&_i_.." has been remembered as 
"&&result&_i_..". ;
        %put %str(     ) This value is stored in the global macro variable 
"&&string&_i_..". ;
        %if %sysevalf(%superq(options_id)=,boolean) = 0 %then %do;
          %put %str(     ) This value is remembered with an options ID stored 
in the global macro variable "&&string_id&_i_..". ;
        %end;
      %end;

    %end;
    %else %do;

      %*** If the put flag is used, then put information to the log. ;
      %if %sysfunc(upcase(&put.)) = Y %then %do;
        %put INFO: Option "&&option&_i_.." has already been remembered with 
initial value "&&&&&&string&_i_...". ;
        %put %str(     ) This value is stored in the global macro variable 
"&&string&_i_..". ;
        %put %str(     ) To change the remembered value of "&&option&_i_.." 
the option must first be reset using 
"%nrstr(%%)options_reset(&&option&_i_..)". ;
      %end;

    %end;

  %end;

%mend options_remember;
```

Restricted - External

```
%macro options_reset(options=, options_id=, put=Y) /
des = 'Reset an option to the value that was retained by OPTIONS_REMEMBER.';

  %*** Set macro variables to be local to this macro to avoid conflicts. ;
  %local _i_ _options_ num_options;

  %*** Clean the input of any spurious spaces. ;
  %let _options_=%sysfunc(strip(%sysfunc(compbl(&options.))));

  %*** Count how many options have been passed. ;
  %let num_options=%sysfunc(countw(&_options_.));

  %*** If the put flag is used, then put information to the log. ;
  %if %sysfunc(upcase(&put.)) = Y %then %do;
    %put INFO: There have been %sysfunc(strip(&num_options.)) options
passed.;
    %put INFO: The options that will be recalled are "&_options_.".;
  %end;

  %*** Loop for each option. ;
  %do _i_=1 %to &num_options.;

    %*** Set looped macro variables to be local to this macro to avoid
conflicts. ;
    %local option&_i_. string&_i_. result&_i_.;

    %*** Find the nth option in the string passed. ;
    %let option&_i_.=%sysfunc(scan(&_options_.,&_i_.));

    %*** String will be populated with the option name wrapped with
underscores. ;
    %let string&_i_.=%sysfunc(compress(_&&option&_i_.._));

    %*** Test to see if the string exists as a macro variable. ;
    %let result&_i_.=%symexist(&&string&_i_..);

    %*** If the macro variable does exist then there are 5 possible outcomes.
;
    %if &&result&_i_..=1 %then %do;

      %*** Set additional looped macro variables to be local to this macro to
avoid conflicts. ;
      %local string_id&_i_. result_id&_i_.;

      %*** String id will be populated with the option name with "_id" as a
suffix... ;
      %*** ...and the entire string wrapped with underscores. ;
      %let string_id&_i_.=%sysfunc(compress(_&&option&_i_.._id_));

      %*** Test to see if the string id exists as a macro variable. ;
      %let result_id&_i_.=%symexist(&&string_id&_i_..);

      %*** If the value is tagged with an ID then do... ;
      %if &&result_id&_i_..=1 %then %do;

        %*** Test whether the options id macro variable is populated. ;
        %if %sysevalf(%superq(options_id)=,boolean) = 0 %then %do;
```

Restricted - External

```
          %*** Outcome 1: The option is remembered with an ID and reset using
the same ID. ;
          %if &&&&&&string_id&_i_... = %superq(options_id) %then %do;

            %*** Reset the option to the original value. ;
            options &&&&&&string&_i_...;

            %*** If the put flag is used, then put information to the log. ;
            %if %sysfunc(upcase(&put.)) = Y %then %do;
              %put INFO: Option "&&option&_i_.." has been reset to
"&&&&&&string&_i_...".;
            %end;

            %*** Once the option has been reset, we clear the global macro
the option was stored in. ;
            %symdel &&string&_i_..;
            %symdel &&string_id&_i_..;

          %end;

          %*** Outcome 2: The option is remembered with an ID and reset using
a different same ID. ;
          %else %do;

            %*** If the put flag is used, then put information to the log. ;
            %if %sysfunc(upcase(&put.)) = Y %then %do;
              %put INFO: Option "&&option&_i_.." has not been reset as the
option ids do not match.;
            %end;

          %end;

        %end;

        %*** Outcome 3: The option is remembered with an ID and reset without
an ID. ;
        %else %do;

          %*** If the put flag is used, then put information to the log. ;
          %if %sysfunc(upcase(&put.)) = Y %then %do;
            %put INFO: Option "&&option&_i_.." has not been reset as the
option ids do not match.;
          %end;

        %end;

      %end;

      %*** If the value is not tagged with an ID then do... ;
      %else %do;

        %*** Test whether the options id macro variable is populated. ;
        %*** Outcome 4: The option is NOT remembered with an ID but is reset
with an ID. ;
        %if %sysevalf(%superq(options_id)=,boolean) = 0 %then %do;

          %*** If the put flag is used, then put information to the log. ;
          %if %sysfunc(upcase(&put.)) = Y %then %do;
```

Restricted - External

```
            %put INFO: Option "&&option&_i_.." has not been reset as the
option ids do not match.;
          %end;

        %end;

        %*** Outcome 5: The option is NOT remembered with an ID and is NOT
reset with an ID either. ;
        %else %do;

          %*** Reset the option to the original value. ;
          options &&&&&&string&_i_...;

          %*** If the put flag is used, then put information to the log. ;
          %if %sysfunc(upcase(&put.)) = Y %then %do;
            %put INFO: Option "&&option&_i_.." has been reset to
"&&&&&&string&_i_...".;
          %end;

          %*** Once the option has been reset, we clear the global macro the
option was stored in. ;
          %symdel &&string&_i_..;
        %end;

      %end;

    %end;

    %else %do;

      %*** If the put flag is used, then put information to the log. ;
      %if %sysfunc(upcase(&put.)) = Y %then %do;
        %put INFO: Option "&&option&_i_.." does not have an associated macro
variable available for reset.;
      %end;

    %end;

  %end;

%mend options_reset;
```

Restricted - External

APPENDIX B – MACROS DISCUSSED IN THIS PAPER

```
%macro sys_check_permissions(path=, out_ref=work._check_permissions_output_)
/
des = 'Use sys_ls recursively to check the permissions of a file or
directory.';

  %*** Remember the current value of options prior to changing them. ;
  %options_remember(options=linesize mlogic mprint notes pagesize symbolgen,
options_id=sys_check_permissions, put=N);

  options linesize=256 nomlogic nomprint nonotes pagesize=1024 nosymbolgen;

  %*** Define local macro variable(s). ;
  %local z pos path_part path_dir;

  %*** Delete the temp table(s) that are appended to. ;
  proc datasets lib=work nolist;
    delete
      _check_permissions_base_
      ;
    run;
  quit;

  %do z = 1 %to %eval(%sysfunc(countw(&path.,/))-0);

    %*** Find the positin of each / ;
    data _null_;
      call scan("&path.",&z.,POS,LEN,"/");
      call symputx('pos',POS-2);
    run;

    %let path_part=%substr(&path.,1,&pos.);
    %let path_dir=%scan(&path.,&z.,/);

    %if %sysfunc(fileexist(&path_part.))=1 %then %do;

      %*** List out the current directory / file in the path we are looping
through. ;
      %sys_ls(ls_path=&path_part., out_ref=work._check_permissions_dir_);

      %*** Only keep the relevant member. ;
      data work._check_permissions_line_;
        set work._check_permissions_dir_;
        where scan(FileSpecification,-1,'/')="&path_dir.";
      run;

      %*** Append to a datastore so that we have all rows found. ;
      proc append
        base = work._check_permissions_base_
        data = work._check_permissions_line_;
      run;

    %end;
    %else %do;
      %put INFO: File Specification %superq(path_part) does NOT exist. ;
```

Restricted - External

```
      %goto leave;
    %end;

  %end;

  %*** Give each row a number - this will be from 1 (first directory) to n
(last directory or the file). ;
  data work._check_permissions_n_;
    set work._check_permissions_base_ (keep=GroupName);
    n = _n_;
  run;

  %*** Grouping by group name we can find the top folder in a path where a
group name is used. ;
  proc sql;
  create table work._check_permissions_list_ as
    select
      GroupName,
      min(n) as n
    from work._check_permissions_n_
    group by GroupName
    order by n
  ;quit;

  %*** Count the unique groups. ;
  proc sql noprint;
    select count(*)
    into :group_count
    from work._check_permissions_list_
  ;quit;

  %let group_count = %sysfunc(strip(&group_count.));

  %*** Read the groups into a macro variable array. ;
  proc sql noprint;
    select GroupName
    into:group_name1-:group_name&group_count.
    from work._check_permissions_list_
  ;quit;

  %*** Evaluate the effective permission for others by retaining groups from
one row to the next. ;
  data work._check_permissions_rw_ (drop=GroupNamePrior OtherName
OtherNamePrior );
    set work._check_permissions_base_ (keep=FileSpecification ShortFileName
OwnerName GroupName Type AccessPermission);
    by Type FileSpecification;
    length
      GroupNamePrior $32
      OtherName $32
      OtherNamePrior $32
      %do z=1 %to &group_count.;
        &&group_name&z.. $2
      %end;
      Read $32
      Write $32
      ;
    retain GroupNamePrior OtherNamePrior;
    %*** Test if there are some read or write permission for "other" ;
    if strip(compress(substr(AccessPermission,8,2),'-')) ^= '' then do;
      %*** If the group name is the same for both rows and other has
permissions then other is equal too... ;
      %*** ...but if the group name is different across two rows then this
implies other is the group from the row above. ;
      if GroupName = GroupNamePrior then do;
        OtherName = OtherNamePrior;
      end;
      else if GroupName ^= GroupNamePrior then do;
        OtherName = GroupNamePrior;
      end;
    end;
    else do;
      OtherName = '';
    end;

    %*** Each group will have its own column in the ordered they are
encountered in the path. ;
    %*** The group column will contain any read or write permissions for the
current path. ;
    %do z=1 %to &group_count.;
      if GroupName = "&&group_name&z.." then &&group_name&z.. =
strip(compress(substr(AccessPermission,5,2),'-'));
      else if OtherName = "&&group_name&z.." then &&group_name&z.. =
strip(compress(substr(AccessPermission,8,2),'-'));
    %end;

    GroupNamePrior = GroupName;
    OtherNamePrior = OtherName;

    %*** Determine which is the first column with read (not write) and write
(can be read too) permissions. ;
    %do z=1 %to &group_count.;

      if Read = '' then do;
        if index(&&group_name&z..,'r') > 0 and index(&&group_name&z..,'w') =
0 then do;
          Read = "&&group_name&z..";
        end;
      end;

      if Write = '' then do;
        if index(&&group_name&z..,'r') > 0 and index(&&group_name&z..,'w') >
0 then do;
          Write = "&&group_name&z..";
        end;
      end;

    %end;

    %*** If we are not the last directory / file then we dont need write so
read is write until it actually matters. ;
    if Write = '' then do;
      if not last.type then do;
        Write = Read;
      end;
```

Restricted - External

```
    end;
  run;

  %*** Count how many of the folders have a permission route compatible with
read and write. ;
  proc sql noprint;
    select
      count(*) as row_count,
      count(read) as read_count,
      count(write) as write_count
    into
      :row_count,
      :read_count,
      :write_count
    from work._check_permissions_rw_
  ;quit;

  %*** Get the output names shorter for the sys_log_print macro call. ;
  proc sql;
  create table work._check_permissions_print_ as
    select
      FileSpecification,
      OwnerName as Owner,
      GroupName as Group,
      AccessPermission as Permissions
    from work._check_permissions_rw_
  ;quit;

  %*** Store just the last row in a dataset. ;
  data work._check_permissions_last_;
    set work._check_permissions_print_ nobs=nobs;
    if _n_ = nobs then output;
  run;

  %*** Store certain fields in macro variables for outputting to the log when
there are issues. ;
  proc sql noprint;
    select
      FileSpecification,
      Owner,
      Permissions
    into
      :FileSpecification,
      :Owner,
      :Permissions
    from work._check_permissions_last_
  ;quit;

  %*** Print the output to the log in a way that is easily readable. ;
  %sys_log_print(in_ref=work._check_permissions_print_);

  %*** If all rows have a read value then print the effective read path. ;
  %if &row_count. = &read_count. %then %do;

    proc sql;
    create table work._check_permissions_read_ (keep=Group) as
      select distinct
        b.GroupName as Group,
```

Restricted - External

```
        b.n
      from work._check_permissions_rw_ as a
      inner join work._check_permissions_list_ as b
        on a.Read = b.GroupName
      order by n
    ;quit;

    %put INFO: To Read from "%sysfunc(strip(%superq(FileSpecification)))" you 
need the following groups: ;

    %*** Print the output to the log in a way that is easily readable. ;
    %sys_log_print(in_ref=work._check_permissions_read_);

  %end;
  %else %do;
    %put INFO: Unable to READ from 
"%sysfunc(strip(%superq(FileSpecification)))" ;
    %put INFO: Please contact the owner "%sysfunc(strip(%superq(Owner)))" 
regarding permissions "%sysfunc(strip(%superq(Permissions)))";
    %put ;
  %end;

  %*** If all rows have a write value then print the effective write path. ;
  %if &row_count. = &write_count. %then %do;

    proc sql;
    create table work._check_permissions_write_ (keep=Group) as
      select distinct
        b.GroupName as Group,
        b.n
      from work._check_permissions_rw_ as a
      inner join work._check_permissions_list_ as b
        on a.Write = b.GroupName
      order by n
    ;quit;

    %put INFO: To WRITE to "%sysfunc(strip(%superq(FileSpecification)))" you 
need the following groups: ;

    %*** Print the output to the log in a way that is easily readable. ;
    %sys_log_print(in_ref=work._check_permissions_write_);

  %end;
  %else %do;
    %put INFO: Unable to WRITE to 
"%sysfunc(strip(%superq(FileSpecification)))" ;
    %put INFO: Please contact the owner "%sysfunc(strip(%superq(Owner)))" 
regarding permissions "%sysfunc(strip(%superq(Permissions)))";
    %put ;
  %end;

  %*** Create the output dataset related to out ref with appropriate columns 
that wont confuse the users. ;
  proc sql;
  create table %superq(out_ref) as
    select *
    from work._check_permissions_rw_ (drop=read write)
  ;quit;
  %leave:

  %*** Delete the temp table(s). ;
  proc datasets lib=work nolist;
    delete
      _check_permissions_base_
      _check_permissions_dir_
      _check_permissions_line_
      _check_permissions_n_
      _check_permissions_list_
      _check_permissions_rw_
      _check_permissions_print_
      _check_permissions_last_
      _check_permissions_max_
      _check_permissions_read_
      _check_permissions_write_
      ;
    run;
  quit;

  %options_reset(options=linesize mlogic mprint notes pagesize symbolgen,
options_id=sys_check_permissions, put=N);

%mend sys_check_permission;
```

```
%macro sys_dir_create(path=, put=Y) /
des = 'Use the dcreate function to recursively create a directory on the
system.';

  %*** Make the macro_msg macro variable available globally ;
  %global macro_msg;

  data _null_;
    length
      new_path $512
      ;

    %*** Count the number of occurrences of the directory break character ;
    dirn = countw("%superq(path)",'/');

    %*** Loop for each directory ;
    do i=1 to dirn;
      call scan("%superq(path)", i, POS, LEN, "/");
      %*** Define the full path of the parent directory ;
      if POS-2 > 0
        then parent_path = substr("%superq(path)", 1, POS-2);
        else parent_path = '/';

      %*** Define the name of the child directory (without full path) ;
      dir = scan("%superq(path)", i, "/");

      %*** Define the full path of the child directory ;
      if parent_path = '/'
        then child_path = cats(parent_path,dir);
        else child_path = cats(parent_path,'/',dir);

      %*** Test whether the parent and child paths exist ;
      parent_exist = fileexist(parent_path);
      child_exist = fileexist(child_path);
      %*** If the parent directory exists but the child directory does not
then do... ;
      if parent_exist = 1 and child_exist = 1 then do;
        if "%superq(put)" = 'Y' then do;
          put 'Directory already exists: ' child_path;
        end;
      end;
      else if parent_exist = 1 and child_exist = 0 then do;
        if "%superq(put)" = 'Y' then do;
          put 'Directory does not exist. Creating now: ' child_path;
        end;

        %*** Step 1 : Perform command and get return code ;
        new_path = dcreate(dir, parent_path);

        %*** Step 2 : Get system message ;
        macro_msg = sysmsg();

        %*** Step 3 : Put the return code and system message into macro
variables ;
        call symput('macro_msg',strip(macro_msg));

      end;
      else if parent_exist = 0 then do;
```

Restricted - External

```
        if "%superq(put)" = 'Y' then do;
          put 'Parent directory does not exist. Cannot create: ' parent_path;
        end;
      end;
    end;
  run;

  %*** If the put flag is used, then put information to the log. ;
  %if %sysfunc(upcase(%superq(put))) = Y %then %do;
    %put %superq(macro_msg) ;
  %end;

%mend sys_dir_create;
```

Restricted - External

```
%macro sys_move(path=, newpath=, put=) /

des = 'Use the rename function to rename or move a file on the system.';

  %*** Make the macro_rc and macro_msg macro variables available globally ;
  %global macro_rc macro_msg;

  data _null_;
    %*** Step 1 : Perform command and get return code ;
    macro_rc = rename("%superq(path)", "%superq(newpath)", 'file');

    %*** Step 2 : Get system message ;
    macro_msg = sysmsg();

    %*** Step 3 : Put the return code and system message into macro variables
;
    call symput('macro_rc',strip(put(macro_rc,8.)));
    call symput('macro_msg',strip(macro_msg));
  run;

  %*** If the put flag is used, then put information to the log. ;
  %if %sysfunc(upcase(%superq(put))) = Y %then %do;
    %put INFO: The macro return code is &macro_rc. ;
    %put %superq(macro_msg) ;
  %end;

%mend sys_move;
```

Restricted - External

```
%macro sys_file_copy(path=, newpath=, put=Y) /
des = 'Use the fread, fget, fput and fwrite functions to copy a file on the
system.';

  %*** Make the macro_rc and macro_msg macro variables available globally ;
  %global macro_rc macro_msg;

  %*** Assign the source and target files ;
  filename _in_ "%superq(path)";
  filename _out_ "%superq(newpath)";

  %*** Copy the file byte-for-byte ;
  data _null_;
    length filein 8 fileout 8;

    %*** Step 1 : Perform commands and get the return code ;
    filein = fopen('_in_','I',1,'B');
    fileout = fopen('_out_','O',1,'B');
    rec = '20'x;
    do while(fread(filein)=0);
       macro_rc = fget(filein,rec,1);
       macro_rc = fput(fileout,rec);
       macro_rc = fwrite(fileout);
    end;
    macro_rc = fclose(filein);
    macro_rc = fclose(fileout);

    %*** Step 2 : Get system message ;
    macro_msg = sysmsg();

    %*** Step 3 : Put the return code and system message into macro variables
;
    call symput('macro_rc',strip(put(macro_rc,8.)));
    call symput('macro_msg',strip(macro_msg));
  run;

  %*** Clear the file assignments ;
  filename _in_ clear;
  filename _out_ clear;

  %*** If the put flag is used, then put information to the log. ;
  %if %sysfunc(upcase(%superq(put))) = Y %then %do;
    %put INFO: The macro return code is &macro_rc. ;
    %put %superq(macro_msg) ;
  %end;

%mend sys_file_copy;
```

Restricted - External

```
%macro sys_delete(path=, put=Y) /

des = 'Use the fdelete function to delete a file or dir on the system.';

  %*** Make the macro_rc and macro_msg macro variables available globally ;
  %global macro_rc macro_msg;

  %*** Assign a file that you want to delete ;
  filename _in_ "%superq(path)";

  data _null_;
    %*** Step 1 : Perform command and get return code ;
    macro_rc = fdelete('_in_');

    %*** Step 2 : Get system message ;
    macro_msg = sysmsg();

    %*** Step 3 : Put the return code and system message into macro variables
;
    call symput('macro_rc',strip(put(macro_rc,8.)));
    call symput('macro_msg',strip(macro_msg));
  run;

  %*** Clear the file assignment ;
  filename _in_ clear;

  %*** If the put flag is used, then put information to the log. ;
  %if %sysfunc(upcase(%superq(put))) = Y %then %do;
    %put INFO: The macro return code is &macro_rc. ;
    %put %superq(macro_msg) ;
  %end;

%mend sys_delete;
```

Restricted - External

```
%macro sys_find(find_path=, max_depth=, format=L, out_ref=work._find_output_,
put=N) /
des = 'Use various functions to mimic the recursive listing of a directory on
the system.';

  %*** Remember the current value of options prior to changing them. ;
  %options_remember(options=mlogic mprint notes symbolgen,
options_id=sys_find, put=N);

  options nomlogic nomprint nosymbolgen nonotes;

  %*** Define local macro variable(s). ;
  %local dsopt where_max_depth dir_count loop_count directory depth ;

  %*** Dataset options that allow the process to run faster. ;
  %let dsopt = COMPRESS=NO;

  %*** Determine and handle if the max_depth parameter is being used. ;
  %if %sysfunc(compress(%superq(max_depth),,kd)) = %superq(max_depth)
  and %sysevalf(%superq(max_depth)=,boolean) = 0 %then %do;
    %let where_max_depth = where depth le %superq(max_depth);
  %end;
  %else %do;
    %let where_max_depth = ;
  %end;

  %*** Clear down any pre-existing work tables that we append to. ;
  proc datasets lib=work nolist;
    delete
      _FIND_CHECKED_
      _FIND_LISTING_
      _FIND_UNCHECKED_
      ;
    run;
  quit;

  %*** Create the base unchecked table using the find path. ;
  data WORK._FIND_UNCHECKED_ (&dsopt.);
    length FileSpecification $512 Depth 8.;
    if length("%superq(find_path)") = 1 or
substr("%superq(find_path)",length("%superq(find_path)"),1) ^= '/' then do;
      FileSpecification = "%superq(find_path)";
    end;
    else do;
      FileSpecification =
substr("%superq(find_path)",1,length("%superq(find_path)")-1);
    end;
    Depth = 0;
  run;

  %*** Set a default value of the Directory Count to 1 (because we start with
1 directory). ;
  %let dir_count = 1;
  %let loop_count = 0;

  %*** 1.0.4 - If the put flag is used, then put information to the log. ;
  %if %sysfunc(upcase(%superq(put))) = Y %then %do;
    %put ;
    %put Start of find loop...;
    %put ;
  %end;

  %*** Continue to loop as long as there are additional directories to
explore. ;
  %do %while (&dir_count. > 0);

    %*** Define the first row as the next directory to examine. ;
    %*** All other rows remain in the unchecked directory list (grows as
process loops). ;
    data
      WORK._FIND_NEXT_ (&dsopt.)
      WORK._FIND_UNCHECKED_ (&dsopt.)
      ;
      set WORK._FIND_UNCHECKED_;

      if _n_ = 1
        then output _FIND_NEXT_;
        else output _FIND_UNCHECKED_;
    run;

    %*** Put the values for the current directory into macro variables. ;
    proc sql noprint;
      select
        FileSpecification,
        Depth
      into
        :Directory,
        :Depth
      from WORK._FIND_NEXT_
    ;quit;

    %let Loop_Count = %eval(&loop_count. + 1);

    %*** 1.0.4 - If the put flag is used, then put information to the log. ;
    %if %sysfunc(upcase(%superq(put))) = Y %then %do;
      %put &loop_count. : %superq(directory);
    %end;

    %*** Perform a listing on the current directory. ;
    %sys_ls(ls_path=%superq(directory),
format=%sysfunc(upcase(%superq(format))));

    %*** Append the directory that was just checked into the checked list. ;
    proc append
      base = WORK._FIND_CHECKED_
      data = WORK._FIND_NEXT_;
    run;

    %*** Define the depth for new items being added to the list of found
objects. ;
    proc sql;
    create table WORK._FIND_NEW_LISTING_ as
      select a.*
      from
        (
        select *,
```

Restricted - External

```
        case
          when FileSpecification = "%superq(directory)"
            then &depth.
            else &depth. + 1
        end as Depth
      from WORK._LS_OUTPUT_
      ) as a
    %superq(where_max_depth)
  ;quit;

  %*** Append the new objects to the overall list. ;
  proc append
    base = WORK._FIND_LISTING_
    data = WORK._FIND_NEW_LISTING_;
  run;

  %*** Define the items which we need to add to the unchecked list. ;
  proc sql;
  create table WORK._FIND_NEW_UNCHECKED_ as
    select
      a.FileSpecification,
      a.Depth
    from WORK._FIND_NEW_LISTING_ as a
    left join WORK._FIND_CHECKED_ as b
      on a.FileSpecification = b.FileSpecification
    left join WORK._FIND_UNCHECKED_ as c
      on a.FileSpecification = c.FileSpecification
    where a.Type = 'd'
      and b.FileSpecification = ''
      and c.FileSpecification = ''
  ;quit;

  %*** Append the items to the unchecked list. ;
  proc append
    base = WORK._FIND_UNCHECKED_
    data = WORK._FIND_NEW_UNCHECKED_;
  run;

  %*** Count how many directories are left to be checked. ;
  proc sql noprint;
    select count(*) into: Dir_Count
    from WORK._FIND_UNCHECKED_
  ;quit;

%end;

%*** 1.0.4 - If the put flag is used, then put information to the log. ;
%if %sysfunc(upcase(%superq(put))) = Y %then %do;
  %put ;
  %put End of find loop...;
  %put ;
%end;

%*** Create a final ordered output with distinct rows. ;
proc sql;
create table %superq(out_ref) as
  select distinct *
  from WORK._FIND_LISTING_
```

Restricted - External

```
    order by FileSpecification
  ;quit;

  %*** Delete the temp table(s). ;
  proc datasets lib=work nolist;
    delete
      _LS_OUTPUT_
      _FIND_NEXT_
      _FIND_CHECKED_
      _FIND_NEW_LISTING_
      _FIND_LISTING_
      _FIND_NEW_UNCHECKED_
      _FIND_UNCHECKED_
      ;
    run;
  quit;

  %*** Reset the options to their original stored values. ;
  %options_reset(options=mlogic mprint notes symbolgen, options_id=sys_find,
put=N);

%mend sys_find;
```

Restricted - External

```
%macro sys_ls(ls_path, format=L, out_ref=work._ls_output_) /
des = 'Use various functions to mimic the listing of a directory on the
system.';

  %*** Remember the current value of options prior to changing them. ;
  %options_remember(options=mlogic mprint notes symbolgen, options_id=sys_ls,
put=N);

  options nomlogic nomprint nonotes nosymbolgen;

  %*** Define local macro variable(s). ;
  %local keep dsopt;

  %*** Dataset options that allow the process to run faster. ;
  %let dsopt = COMPRESS=NO;

  %*** Create the base transient table using the ls_path. ;
  data WORK._LS_NEXT_ (&dsopt.);
    length FileSpecification $512;
    if length("%superq(ls_path)") = 1 or
substr("%superq(ls_path)",length("%superq(ls_path)"),1) ^= '/' then do;
      FileSpecification = "%superq(ls_path)";
    end;
   else do;
      FileSpecification =
substr("%superq(ls_path)",1,length("%superq(ls_path)")-1);
    end;
  run;

  %*** In order to avoid file-lock errors in log, route log to black hole. ;
  proc printto log="/dev/null";
  run;

  %*** List all child files and child directories within a parent directory.
;
  data
    WORK._LS_LISTING_
      (keep=
        FileSpecification
        MemberName
        OwnerName
        GroupName
        AccessPermission
        FileSize
        LastModified
        Readable
        SysMsg
      &dsopt.)
    ;
    set WORK._LS_NEXT_ (keep=FileSpecification);

    length
      pref $8
      cref $8
      OwnerName $32
      GroupName $32
      AccessPermission $12
      Filesize 8.
      LastModified $48
      Readable $1
      SysMsg $512
      ;

    %*** Assign a file reference to the parent directory. ;
    rc1 = filename(pref, FileSpecification);

    %*** If the file assignment was successful then try and open the parent
directory. ;
    %*** (NO)TE: File assignments are often successful in undesirable
situations. ;
    if rc1 = 0 then do;
      pid = dopen(pref);
      %*** Store any system message i.e. ;
      %*** (ER)ROR: Physical file does not exist, /.../.../... ;
      %*** (ER)ROR: Insufficient authorization to access /.../.../... ;
      SysMsg = sysmsg();
    end;
    else do;
      pid = .;
    end;

    %*** Clear the file assignment as we are now using the pid variable. ;
    rc2 = filename(pref);

    %*** If opening the parent directory was unsuccessful then output. ;
    if pid <= 0 then do;

      %*** ------------------------------------------------------------------
--;
      %*** OUTPUT BLOCK 1 - Failed to read parent directory.
;
      %*** ------------------------------------------------------------------
--;
      OwnerName = '';
      GroupName = '';
      AccessPermission = '';
      FileSize = .;
      LastModified = '';
      Readable = 'N';
      output;

    end;
    %*** If opening the parent directory was successful then gather
additional information. ;
    else do;

      %*** ------------------------------------------------------------------
--;
      %*** OUTPUT BLOCK 2 - Successfully read parent directory.
;
      %*** ------------------------------------------------------------------
--;
      OwnerName = dinfo(pid,'Owner Name');
      GroupName = dinfo(pid,'Group Name');
      AccessPermission = dinfo(pid,'Access Permission');
      FileSize = .;
```

Restricted - External

```
      LastModified = dinfo(pid,'Last Modified');
      Readable = 'Y';
      output;

      %*** Determine the number of child members in the parent directory. ;
      dnum = dnum(pid);

      %*** Loop for each child member of the parent directory. ;
      do i = 1 to dnum;

        %*** Determine the name of the current child member (file or
directory). ;
        MemberName = dread(pid, i);

        %*** Define the file specification using the parent directory and
child member name. ;
        if dinfo(pid,'Directory') = '/' then do;
          FileSpecification = cats(dinfo(pid,'Directory'),MemberName);
        end;
        else do;
          FileSpecification = cats(dinfo(pid,'Directory'),'/',MemberName);
        end;

        %*** Ignore file locks. ;
        if index(MemberName,'.lck') = 0 then do;

          %*** Assign a file reference to the child directory. ;
          rc3 = filename(cref, FileSpecification);

          %*** If the file assignment was successful then try and open the
child directory. ;
          %*** (NO)TE: File assignments are often successful in undesirable
situations. ;
          if rc3 = 0 then do;
            cid = dopen(cref);
            %*** Store any system message i.e. ;
            %*** (ER)ROR: Physical file does not exist, /.../.../... ;
            %*** (ER)ROR: Insufficient authorization to access /.../.../... ;
            SysMsg = sysmsg();
          end;
          else do;
            cid = .;
          end;

          %*** Clear the file assignment as we are now using the cid
variable. ;
          rc4 = filename(cref);

          %*** Override errors for trying to open a member as a directory
when it is not a directory. ;
          if substr(SysMsg,1,length('ERROR: A component of')) = 'ERROR: A
component of'
          and substr(SysMsg,length(SysMsg)-length('is not a
directory.')+1,length('is not a directory.')) = 'is not a directory.' then
do;
            SysMsg = '';
          end;
```

Restricted - External

```
          %*** If opening the child directory was successful then output. ;
          if cid > 0 then do;

            %*** ------------------------------------------------------------
--------;
            %*** OUTPUT BLOCK 3 - Successfully read child directory.
;
            %*** ------------------------------------------------------------
--------;
            OwnerName = dinfo(cid,'Owner Name');
            GroupName = dinfo(cid,'Group Name');
            AccessPermission = dinfo(cid,'Access Permission');
            FileSize = .;
            LastModified = dinfo(cid,'Last Modified');
            Readable = 'Y';
            output;

          end;
          %*** Else try and handle the object as a file. ;
          else if upcase("%superq(format)") = 'L' then do;

            %*** Attempt to open the child member as a file. ;
            fid = mopen(pid, MemberName, 'I');

            %*** Store any system message. ;
            SysMsg = sysmsg();

            %*** If the child member is a file and it opened successfully
then do this. ;
            if fid > 0 then do;

              %*** ----------------------------------------------------------
----------;
              %*** OUTPUT BLOCK 4 - Successfully read child file.
;
              %*** ----------------------------------------------------------
----------;
              OwnerName = finfo(fid,'Owner Name');
              GroupName = finfo(fid,'Group Name');
              AccessPermission = finfo(fid,'Access Permission');
              FileSize = input(finfo(fid,'File Size (bytes)'),32.);
              LastModified = finfo(fid,'Last Modified');
              Readable = 'Y';
              output;

            end;
            else if substr(SysMsg,1,length('ERROR: File is in use')) =
'ERROR: File is in use' then do;

              %*** ----------------------------------------------------------
----------;
              %*** OUTPUT BLOCK 5 - Child member is in use and cant be read.
;
              %*** ----------------------------------------------------------
----------;
              OwnerName = '';
              GroupName = '';
              AccessPermission = '';
              FileSize = .;
              LastModified = '';
              Readable = 'X';
              output;

            end;
            else do;

              %*** ------------------------------------------------------------
----------;
              %*** OUTPUT BLOCK 6 - Child member has an error other than cant
be read. ;
              %*** ------------------------------------------------------------
----------;
              OwnerName = '';
              GroupName = '';
              AccessPermission = '';
              FileSize = .;
              LastModified = '';
              Readable = 'N';
              output;

            end;
          end;
          else do;

            %*** --------------------------------------------------------------
--------;
            %*** OUTPUT BLOCK 7 - For short listing only.
;
            %*** --------------------------------------------------------------
--------;
            OwnerName = '';
            GroupName = '';
            AccessPermission = '';
            FileSize = .;
            LastModified = '';
            Readable = 'Y';
            output;

          end;

          %*** Close the child directory. ;
          rc5 = dclose(cid);

        end;
      end;
    end;

    %*** Close the parent directory. ;
    rc6 = dclose(pid);
  run;

  %*** Get the log back. ;
  proc printto log=LOG;
  run;

  %*** Tidy up variables and define new ones where required. ;
  data WORK._LS_CLEANSING_;
    set WORK._LS_LISTING_;
    length
      Type $1
      OctalPermission 3
      LastModifiedDate 8
      LastModifiedTime 8
      ;
    format
      LastModifiedDate date9.
      LastModifiedTime time8.
      ;
    %*** Define the type based on the first substring of the access
permissions. ;
    if substr(AccessPermission,1,1) ^= '-'
      then Type = substr(AccessPermission,1,1);
      else Type = 'f';

    %*** Define the octal representation of the access permissions. ;
    OctalPermission = 0;
    if AccessPermission ^= '' then do;
      if substr(AccessPermission,2,1) ^= '-' then OctalPermission =
OctalPermission + 400;
      if substr(AccessPermission,3,1) ^= '-' then OctalPermission =
OctalPermission + 200;
      if substr(AccessPermission,4,1) ^= '-' then OctalPermission =
OctalPermission + 100;
      if substr(AccessPermission,5,1) ^= '-' then OctalPermission =
OctalPermission + 40;
      if substr(AccessPermission,6,1) ^= '-' then OctalPermission =
OctalPermission + 20;
      if substr(AccessPermission,7,1) ^= '-' then OctalPermission =
OctalPermission + 10;
      if substr(AccessPermission,8,1) ^= '-' then OctalPermission =
OctalPermission + 4;
      if substr(AccessPermission,9,1) ^= '-' then OctalPermission =
OctalPermission + 2;
      if substr(AccessPermission,10,1) ^= '-' then OctalPermission =
OctalPermission + 1;
    end;
    else do;
      OctalPermission = .;
    end;

    %*** If there is a last modified value then define the last modified date
and time. ;
    if LastModified ^= '' then do;
      LastModifiedDate = input(cats(scan(LastModified,1,'
'),substr(scan(LastModified,2,' '),1,3),scan(LastModified,3,' ')),date9.);
      LastModifiedTime = input(scan(LastModified,4,' '),time8.);
    end;
  run;

  %*** Define which variables we will keep based on the long or short
formats. ;
  %if %sysfunc(upcase(%superq(format))) = L %then %do;
    %let keep = OwnerName, GroupName, Type, AccessPermission,
OctalPermission, FileSize, LastModifiedDate, LastModifiedTime;
```

```
  %end;
  %else %do;
    %let keep = Type;
  %end;

  %*** Re-order the columns and order as desired. ;
  proc sql;
  create table %superq(out_ref) as
    select
      FileSpecification,
      case
        when FileSpecification = '/'
          then ''
        when count(FileSpecification,'/') = 1
          then '/'
        else substr(FileSpecification,1,length(FileSpecification)-
length(scan(FileSpecification,-1,'/'))-1)
      end as ParentDirectory length=512,
      case
        when FileSpecification = '/'
          then ''
        else scan(FileSpecification,-1,'/')
      end as ShortFileName length=128,
      %superq(keep),
      Readable,
      SysMsg
    from WORK._LS_CLEANSING_
    order by FileSpecification
  ;quit;

  %*** Delete the temp table(s). ;
  proc datasets lib=work nolist;
    delete
      _LS_NEXT_
      _LS_LISTING_
      _LS_CLEANSING_
      ;
    run;
  quit;

  %*** Remember the current value of options prior to changing them. ;
  %options_reset(options=mlogic mprint notes symbolgen, options_id=sys_ls,
put=N);

%mend sys_ls;
```

```
%macro sys_zip_contents(zip_fs=, type=DATA,
out_ref=work._zip_contents_output_);

  %put ;
  %put INFO: zip_fs    = %superq(zip_fs);
  %put INFO: type      = %superq(type);
  %put INFO: out_ref   = %superq(out_ref);

  %*** Assign the location of the zip file. ;
  filename zip_fs zip "%superq(zip_fs)";

  %*** Check 1a: The zip file exists. ;
  %if %sysfunc(fexist(zip_fs)) = 1 %then %do;
    %put CHECK: The zip file exists. ;

    %*** If any of the correct types are passed then gather the contents
information. ;
    %if %sysfunc(upcase(%superq(type))) = DATA or
%sysfunc(upcase(%superq(type))) = LOG or %sysfunc(upcase(%superq(type))) =
OUTPUT %then %do;

      %*** Read the list of members from the zip file. ;
      data work._zip_contents_ (keep=memname);
        length memname $512;
        fid=dopen("zip_fs");
        if fid=0 then
          stop;
        memcount=dnum(fid);
        do i=1 to memcount;
          memname=dread(fid,i);
          output;
        end;
        rc=dclose(fid);
      run;

    %end;

    *** If the type is DATA then output to the specified output reference. ;
    %if %sysfunc(upcase(%superq(type))) = DATA %then %do;
      data %superq(out_ref);
        set work._zip_contents_;
      run;
    %end;

    %*** If the type is LOG then print the information to the log. ;
    %else %if %sysfunc(upcase(%superq(type))) = LOG %then %do;
      data _null_;
        set work._zip_contents_;
        put (_all_)( = );
      run;
    %end;

    %*** If the type is OUTPUT then print the information to the results /
output window. ;
    %else %if %sysfunc(upcase(%superq(type))) = OUTPUT %then %do;
      proc print
        data = work._zip_contents_;
        var _all_;
      run;
    %end;

    %*** Otherwise put an (ER)ROR to the log. ;
    %else %do;
      %put %str(ER)ROR: The only valid values for "type" are "DATA", "LOG"
and "OUTPUT";
    %end;

  %end;

  %*** Check 1b: The zip file does NOT exist. ;
  %else %do;
    %put CHECK: The zip file does NOT exist. ;
  %end;

  %if %sysfunc(exist(work._zip_contents_)) %then %do;

    %*** Delete the temp table(s). ;
    proc datasets lib=work nolist;
      delete
        _zip_contents_
        ;
      run;
    quit;

  %end;

  %*** Clear the file assignments ;
  filename zip_fs clear;

%mend sys_zip_contents;
```

Restricted - External

```
%macro sys_zip_member_add(fs=, zip_fs=, zip_mem=, overwrite=N, delete=N);

  %put ;
  %put INFO: fs        = %superq(fs);
  %put INFO: zip_fs    = %superq(zip_fs);
  %put INFO: zip_mem   = %superq(zip_mem);
  %put INFO: overwrite = %superq(overwrite);
  %put INFO: delete    = %superq(delete);

  %*** Set macro variables to be local to this macro to avoid conflicts. ;
  %local exe_cmd zip_dir;

  %*** Give execute command a default value of 1. ;
  %let exe_cmd = 1;

  %*** Define the parent directory where the zip folder is or will be
created. ;
  %let zip_dir =
%sysfunc(substr(%superq(zip_fs),1,%sysfunc(length(%superq(zip_fs)))-
%sysfunc(length(%sysfunc(scan(%superq(zip_fs),-1,"/"))))-1));

  %*** Assign the current location of the file. ;
  filename fs "%superq(fs)";

  %*** Assign the location of the parent directory where the zip folder is or
will be created. ;
  filename zip_dir "%superq(zip_dir)";

  %*** Assign the location of the zip file. ;
  filename zip_fs "%superq(zip_fs)";

  %*** Assign the location of the zip file and name of the member. ;
  filename zip_mem zip "%superq(zip_fs)" member="%superq(zip_mem)";

  %*** Check 1a: The file does exist and the zip parent directory does exist.
;
  %if %sysfunc(fexist(fs)) = 1 and %sysfunc(fexist(zip_dir)) = 1 %then %do;
    %put CHECK: The file does exist and the zip parent directory does exist.
;
    %*** Check 2a: The overwrite option is set to N which requires further
checks. ;
    %if %sysfunc(upcase(%superq(overwrite))) = N %then %do;
      %put CHECK: The overwrite option is set to N which requires further
checks. ;
      %*** Check 3a: The zip file already exists which requires further
checks. ;
      %if %sysfunc(fexist(zip_fs)) = 1 %then %do;
        %put CHECK: The zip file already exists which requires further
checks. ;
        %*** Check 4a: The zip member already exists and overwrite is
preventing replacement. ;
        %if %sysfunc(fexist(zip_mem)) = 1 %then %do;
          %let exe_cmd = 0;
          %put CHECK: The zip member already exists and overwrite flag is
preventing replacement. ;
        %end;
        %*** Check 4b: The zip member does not already exist. ;
        %else %do;
```

```
          %put CHECK: The zip member does not already exist.  ;
        %end;
      %end;
      %*** Check 3b: The zip file does not already exist. ;
      %else %do;
        %put CHECK: The zip file does not already exist. ;
      %end;
    %end;
    %*** Check 2b: The overwrite option is set to Y which requires no further
checks. ;
    %else %if %sysfunc(upcase(%superq(overwrite))) = Y %then %do;
      %put CHECK: The overwrite option is set to Y which requires no further
checks. ;
    %end;
    %*** Check 2c: The overwrite option is not set to Y or N. ;
    %else %do;
      %let exe_cmd = 0;
      %put CHECK: The overwrite option is not set to Y or N. ;
    %end;
  %end;
  %*** Check 1b: The file does not exist and the zip parent directory does
exist. ;
  %else %if %sysfunc(fexist(fs)) = 0 and %sysfunc(fexist(zip_dir)) = 1 %then
%do;
    %let exe_cmd = 0;
    %put CHECK: The file does not exist and the zip parent directory does
exist. ;
  %end;
  %*** Check 1c: The file does exist and the zip parent directory does not
exist. ;
  %else %if %sysfunc(fexist(fs)) = 1 and %sysfunc(fexist(zip_dir)) = 0 %then
%do;
    %let exe_cmd = 0;
    %put CHECK: The file does exist and the zip parent directory does not
exist. ;
  %end;
  %*** Check 1d: The file does not exist and the zip parent directory does
not exist. ;
  %else %if %sysfunc(fexist(fs)) = 0 and %sysfunc(fexist(zip_dir)) = 0 %then
%do;
    %let exe_cmd = 0;
    %put CHECK: The file does not exist and the zip parent directory does not
exist. ;
  %end;

  %*** Outcome 1a: Command will execute. ;
  %if %superq(exe_cmd) = 1 %then %do;

    %*** Copy the file into the zip file member byte-by-byte ;
    data _null_;
      infile fs recfm=n;
      file zip_mem recfm=n;
      input byte $char1. @;
      put byte $char1. @;
    run;

    %*** Outcome 2a: Member has been successfully added to zip file. ;
    %if %sysfunc(fexist(zip_mem)) = 1 %then %do;
```

```
      %put OUTCOME: Member has been added to zip file. ;

      %*** Only do the next section if the macro is set to delete the
original after adding the member. ;
      %if %sysfunc(upcase(%superq(delete))) = Y %then %do;
        data _null_;
          rc = fdelete('fs');
          if rc=0 then do;
            put "DELETE: Successfully deleted %superq(fs).";
          end;
          else do;
            put "DELETE: Unable to delete %superq(fs).";
          end;
        run;
      %end;
      %else %do;
        %put DELETE: Delete flag is preventing deletion of %superq(fs).;
      %end;

    %end;
    %*** Outcome 2b: Member has NOT been successfully added to zip file. ;
    %else %do;
      %put OUTCOME: Member has NOT been successfully added to zip file. ;
    %end;
  %end;
  %*** Outcome 1b: Command will NOT execute due to failed checks. ;
  %else %do;
    %put OUTCOME: Command will NOT execute due to failed checks. ;
  %end;

  %*** Clear the file assignments ;
  filename fs clear;
  filename zip_dir clear;
  filename zip_fs clear;
  filename zip_mem clear;

%mend sys_zip_member_add;
```

```
%macro sys_zip_member_delete(zip_fs=, zip_mem=);

  %put ;
  %put INFO: zip_fs    = %superq(zip_fs);
  %put INFO: zip_mem   = %superq(zip_mem);

  %*** Assign the location of the zip file. ;
  filename zip_fs "%superq(zip_fs)";

  %*** Specify the location of the zip file and name of the member. ;
  filename zip_mem ZIP "%superq(zip_fs)" member="%superq(zip_mem)";

  %*** Check 1a: The zip file exists. ;
  %if %sysfunc(fexist(zip_fs)) = 1 %then %do;
    %put CHECK: The zip file exists. ;
    %*** Check 2a: The zip member exists. ;
    %if %sysfunc(fexist(zip_mem)) = 1 %then %do;
      %put CHECK: The zip member exists. ;

      %*** Delete the zip member. ;
      data _null_;
        rc = fdelete('zip_mem');
        if rc=0 then do;
          put "DELETE: Successfully deleted zip member %superq(zip_mem).";
        end;
        else do;
          put "DELETE: Unable to delete zip member %superq(zip_mem).";
        end;
      run;

    %end;
    %*** Check 2b: The zip member does NOT exist. ;
    %else %do;
      %put CHECK: The zip member does NOT exist. ;
    %end;
  %end;
  %*** Check 1b: The zip file does NOT exist. ;
  %else %do;
    %put CHECK: The zip file does NOT exist. ;
  %end;

  %*** Clear the file assignments ;
  filename zip_fs clear;
  filename zip_mem clear;

%mend sys_zip_member_delete;
```

Restricted - External

```
%macro sys_zip_member_extract(fs=, zip_fs=, zip_mem=, overwrite=N, delete=N);

  %put ;
  %put INFO: fs        = %superq(fs);
  %put INFO: zip_fs    = %superq(zip_fs);
  %put INFO: zip_mem   = %superq(zip_mem);
  %put INFO: overwrite = %superq(overwrite);
  %put INFO: delete    = %superq(delete);

  %*** Set macro variables to be local to this macro to avoid conflicts. ;
  %local exe_cmd zip_dir;

  %*** Give execute command a default value of 1. ;
  %let exe_cmd = 1;

  %*** Define the parent directory where the zip folder is or will be
created. ;
  %let fs_dir = %sysfunc(substr(%superq(fs),1,%sysfunc(length(%superq(fs)))-
%sysfunc(length(%sysfunc(scan(%superq(fs),-1,"/"))))-1));

  %*** Assign the location of the output directory. ;
  filename fs_dir "%superq(fs_dir)";

  %*** Assign the location of the output file. ;
  filename fs "%superq(fs)";

  %*** Assign the location of the zip file. ;
  filename zip_fs "%superq(zip_fs)";

  %*** Assign the location of the zip file and name of the member. ;
  filename zip_mem zip "%superq(zip_fs)" member="%superq(zip_mem)";

  %*** Check 1a: The output file directory does exist and the zip file does
exist. ;
  %if %sysfunc(fexist(fs_dir)) = 1 and %sysfunc(fexist(zip_fs)) = 1 %then
%do;
    %put CHECK: The output file directory does exist and the zip file does
exist. ;
    %*** Check 2a: The zip member does exist. ;
    %if %sysfunc(fexist(zip_mem)) = 1 %then %do;
      %put CHECK: The zip member does exist. ;
      %*** Check 3a: The overwrite option is set to N which requires further
checks. ;
      %if %sysfunc(upcase(%superq(overwrite))) = N %then %do;
        %put CHECK: The overwrite option is set to N which requires further
checks. ;
        %*** Check 4a: The output file already exists and overwrite is
preventing replacement. ;
        %if %sysfunc(fexist(fs)) = 1 %then %do;
          %let exe_cmd = 0;
          %put CHECK: The output file already exists and overwrite is
preventing replacement. ;
        %end;
        %*** Check 4b: The output file does not already exist. ;
        %else %do;
          %put CHECK: The output file does not already exist. ;
        %end;
      %end;
      %*** Check 3b: The overwrite option is set to Y which requires no
further checks. ;
      %else %if %sysfunc(upcase(%superq(overwrite))) = Y %then %do;
        %put CHECK: The overwrite option is set to Y which requires no
further checks. ;
      %end;
      %*** Check 3c: The overwrite option is not set to Y or N. ;
      %else %do;
        %let exe_cmd = 0;
        %put CHECK: The overwrite option is not set to Y or N. ;
      %end;
    %end;
    %*** Check 2b: The zip member does not exist. ;
    %else %do;
      %let exe_cmd = 0;
      %put CHECK: The zip member does not exist. ;
    %end;
   %end;
  %*** Check 1b: The output file directory does not exist and the zip file
does exist. ;
  %else %if %sysfunc(fexist(fs_dir)) = 0 and %sysfunc(fexist(zip_fs)) = 1
%then %do;
    %let exe_cmd = 0;
    %put CHECK: The output file directory does not exist and the zip file
does exist. ;
  %end;
  %*** Check 1c: The output file directory does exist and the zip file does
not exist. ;
  %else %if %sysfunc(fexist(fs_dir)) = 1 and %sysfunc(fexist(zip_fs)) = 0
%then %do;
    %let exe_cmd = 0;
    %put CHECK: The output file directory does exist and the zip file does
not exist. ;
  %end;
  %*** Check 1d: The output file directory does not exist and the zip file
does not exist. ;
  %else %if %sysfunc(fexist(fs_dir)) = 0 and %sysfunc(fexist(zip_fs)) = 0
%then %do;
    %let exe_cmd = 0;
    %put CHECK: The output file directory does not exist and the zip file
does not exist. ;
  %end;

  %*** Outcome 1a: Command will execute. ;
  %if %superq(exe_cmd) = 1 %then %do;

    %*** Copy the file out of the zip file in blocks. ;
    data _null_;
      infile zip_mem lrecl=256 recfm=F length=length eof=eof unbuf;
      file fs lrecl=256 recfm=N;

      input;
      put _infile_ $varying256. length;

      return;
      eof:
      stop;
    run;
```

Restricted - External

```
    %*** Outcome 2a: Member has been successfully extracted to output file. ;
    %if %sysfunc(fexist(fs)) = 1 %then %do;
      %put OUTCOME: Member has been successfully extracted to output file. ;

      %*** Only do the next section if the macro is set to delete the
original zip member after extracting it. ;
      %if %sysfunc(upcase(%superq(delete))) = Y %then %do;
        %*** Delete the zip member. ;
        data _null_;
          rc = fdelete('zip_mem');
          if rc=0 then do;
            put "DELETE: Successfully deleted zip member %superq(zip_mem).";
          end;
          else do;
            put "DELETE: Unable to delete zip member %superq(zip_mem).";
          end;
        run;
      %end;
      %else %do;
        %put DELETE: Delete flag is preventing deletion of %superq(zip_mem).;
      %end;

    %end;
    %*** Outcome 2b: Member has NOT been successfully extracted to output
file. ;
    %else %do;
      %put OUTCOME: Member has NOT been successfully extracted to output
file. ;
    %end;
  %end;
  %*** Outcome 1b: Command will NOT execute due to failed checks. ;
  %else %do;
    %put OUTCOME: Command will NOT execute due to failed checks. ;
  %end;

  %*** Clear the file assignments ;
  filename fs_dir clear;
  filename fs clear;
  filename zip_fs clear;
  filename zip_mem clear;

%mend sys_zip_member_extract;
```

Restricted - External