Paper #2362-2018

Efficient Use of Disk Space in SAS® Application Programs

Thomas E. Billings, MUFG Union Bank, N.A., San Francisco, California

ABSTRACT

A tutorial on managing disk space for SAS® data sets and files created by or for the SAS system. Basic housekeeping is covered: keep files that are in-use and backup or discard files that are not in use. Backup methods are discussed, including the important question whether the operating system that your SAS site runs on might change in the future, necessitating use of the special transport format for backup files. SAS procedures that are commonly used for disk file management are described: PROC DELETE, DATASETS, and CATALOG. SQL DELETE and SAS DATA step functions for file management are also discussed. File compression is a very important tool for saving disk space, and the SAS features for this are described. Logical deletion of rows in a data set can waste disk space; prototype SAS code to detect files with this condition is supplied in an appendix. Multiple SAS programming techniques that promote efficient use of disk space are described, as well as suggestions for managing the SAS WORK library.

INTRODUCTION: WHY DISK SPACE STILL MATTERS

The cost of computer hardware is being driven downward by technological advances, and consequently disk drives are getting cheaper and faster. The use of commodity hardware is changing the economics of computer systems, allowing larger systems at lower cost. Most of the programmers who read this paper presumably have computers at home, and may use installed or external disk drives that are 1 terabyte (TB) or more. Disk space for PCs is very low cost; a quick search on amazon.com shows low prices for internal/external 1+ TB disk systems.

Unfortunately, disk space is still expensive for large servers that are critical to enterprises. Enterprise disk systems are usually redundant, with the additional requirement to purchase similar disk space for a disaster recovery server. These servers must be housed in protected buildings with backup electric power. An enterprise server also needs admins and other support staff, all of which have associated costs. This infrastructure/overhead significantly increases the total cost of providing disk space in enterprise servers.

The bottom line is that disk space in an enterprise server is still an important commodity, to be used in a prudent and efficient manner. In the sections that follow, we present an overview of the tools and techniques available to SAS® programmers to make efficient and effective use of disk space.

BASIC HOUSEKEEPING: FILE CLEANUP

Does a file or program need to be saved? It may be necessary or appropriate to save a copy of a file, program, or other computer-generated artifact because of:

- Regulatory, legal, or audit requirements;
- Plan or expectation to reuse the items in the future.

Files to be saved can usually be divided into 2 categories:

- in active use now or in the recent past, or likely to be actively used in the near future, and
- files that probably won't be used in the near future, but a copy should be saved.

The first category of files should remain in-place, and the second can be moved to archival or backup storage.

Versioning. Many production systems – sets of SAS programs – run on a schedule that may be daily, weekly, monthly, or other period, and can produce data sets, logs, html output, graph files, spreadsheets, and so on. How many versions of a production run should be kept online?

If a system produces or updates files that are used for historical reporting, then the data set history files must be retained indefinitely or for a prescribed period. If the question is applied to the other types of output (e.g., logs), then the answer will depend on a number of factors:

- Space required for each production run output – e.g.:
  - Are the log files huge?
  - Are large amounts of space required for graphic files?
- Amount of space available on the system
- Reliability of the system – how often it fails
- If the system fails, number of earlier runs needed to diagnose errors.

Review of the above factors, combined with experimentation, can provide an answer to the versioning question: how many versions to save, with earlier versions moved off the server.

Limiting data saved online to a specified period. To save space, relational databases are often managed to keep online only data for a specified period, with older data rolled off to backup. This is similar to the versioning described above, and can be applied to SAS data sets that store history.

Files to discard. Some files can reasonably be discarded: multiple, obsolete versions of a file or program; files with known errors that have been replaced with correct versions; outdated 1-time files or programs; and so on. Finally, for some files or programs, it may be **unclear** whether the item should be saved or discarded. For those, the prudent action is to save the file in backup.

BACKUP FILES & PROGRAMS

Most enterprises have official IT-supported backup systems & processes for production servers. The systems and methods used will vary depending on what your IT department selected as the official method/system. Backup for production servers is standard and a best practice; development & test servers often do **not** support the backup processes used for production.

Given that you have files to be stored as backup or archived, depending on the server the files reside on, you have options:

- Use the official IT-managed backup process – this should be used for all production files, and also for files being saved for regulatory, legal, or audit reasons
- Other backup options may include:
  - enterprise-internal servers (usually Windows; see remarks below re: CEDA),
  - offline systems like highly compressed tape cartridges (often used by IT and may not be available to non-IT users),
  - your own workstation.

Backup to your workstation is **not** recommended for long term storage as your workstation disks may be wiped when you leave the enterprise, and/or your workstation files might not be backed up elsewhere. If you are backing up files from your own PC running SAS University Edition, then you can use USB memory devices for backup if you wish. For all other applications, USB-backup is not recommended and is in fact prohibited (for security reasons) in many enterprises.

File formats. The SAS system supports a number of file types. The primary focus here is on data sets as they are usually the major consumers of disk space (although systems that produce large numbers of graphs will need to manage those). When archiving SAS-related files and programs, a number of options are available, as follows.

- SAS data sets (.sas7bdat files):
  - SAS native file format with optional file compression (binary, character; details below)
  - SAS transport format – via PROC CPORT, SAS data engine XPORT
  - Zip, tar
- Compiled objects - macros, DATA steps, views, etc. – the most portable way to archive these is to archive the source code/programs that produce the objects.
- Formats (compiled) – formats can be converted to/from data sets using the CNTLOUT= and CNTLIN= options of PROC FORMAT or may be migrated as catalogs using PROC CPORT, CIMPORT. Another alternative available in some instances is to archive the source code used to create the format libraries.
- Programs (source code; text files):
  - Zip, tar
  - Git repository (highly compressed, can save multiple versions and history). Useful for source code in text file format. Git is a popular open source, source code management system; it is available for Linux, Unix, Windows; also z/OS running under IBM Unix Systems Services.
  - Note: do **not** use Git to archive SAS Enterprise Guide project .egp files as they are zipped. However, a version of Git is embedded in SAS Enterprise Guide (version 7.1 and later) and can be used inside a project to manage user-written source code.
- Reproducible research files; generally these files can be zipped:
  - SAS: StatTag (creates Microsoft Word files), SASweave, StatRep
  - **R**: RStudio compiled notebooks (html files); R packages
  - Jupyter notebook files (relevant to R, Python, R, Julia, SAS, and other languages)
- Graphics files: zip, tar. If you are using reproducible research methods, saving the code used to produce the graphs (rather than the graphs themselves) may be a space-saving option.
- Miscellaneous files – pdf, rtf, epub, html – use zip or tar.

Think before you archive SAS files & programs. In the future, will your enterprise still be using the same operating system? Is it safe to save SAS data sets in the native operating system (OS) format, or are other measures appropriate?

SAS supports CEDA: cross-environment data access, which allows files on one OS to access/use files written under a different operating system. CEDA applies to files on directory-based operating systems: Windows, Unix, Linux, and Unix System Services (only) on z/OS. See the SAS documentation (URL in references) for details.

- If your enterprise's OS is unlikely to change, then archiving native SAS file formats is relatively safe.
- If the OS is likely to change or if there is significant uncertainty, consider saving the files/catalogs in the special (sequential) ***transport format*** using PROC CPORT.
- For cross-platform compatibility, the approach is, for 2 systems with different OS:
  - Given SAS files written under OS #1, use PROC CPORT to create transport format copies of the SAS files
  - Transfer the transport format files to the other system or to backup

    - Once the target file is on the system running OS #2, use PROC CIMPORT to create SAS files in the native format for OS #2.
- The SAS data engine XPORT provides another method to create files in transport format.
- PROC CPORT is the only way (in SAS) to backup graphic catalogs; PROC COPY does not work for such files.
- Source code can be included in catalogs as file type SOURCE, and can be backed up with PROC CPORT. Git repositories and text-only files are often easier to work with than SOURCE files in a catalog.

SAS TOOLS FOR MANAGING FILES/LIBRARIES

A SAS library is defined via a LIBNAME statement or function invocation. SAS data files and views are stored in the locations/directories defined via a LIBNAME. Other file types may be stored in a LIBNAME directory, including formats, informats (input formats), compiled macros, graphic files, etc. The non-data file types are considered "catalog" files, and if a SAS LIBNAME contains no data files and only catalog files, then it may be referred to as a catalog.

A LIBNAME usually lists only 1 directory but a LIBNAME can point to multiple, concatenated directories. There is a similar CATNAME statement used to support concatenation of SAS catalogs. To avoid confusion, it is recommended that LIBNAMEs point to only a single physical directory for cleanup work.

SAS provides tools to manage data files in libraries, and also to manage files/members of catalogs. Data files usually take up the most space and they are our primary emphasis. Format libraries however can become very large and may need active management.

Encryption and password-protection of files can complicate the management of disk space, as you may need to supply an encryption key or password to delete a file with SAS tools. Use of operating system commands for file management is a work-around for these files.

Managing SAS data sets in a directory

File deletion. PROC DELETE is the simplest way to delete a small number of SAS datasets. Syntax:

```
proc delete data=a.b1 a.b2;
run;
```

where b1, b2 are the names of the files to be deleted and a is the relevant libref. The PROC also works for SAS generation data sets and also encrypted files (the encryption key is provided automatically for metadata-bound libraries in a metadata environment; otherwise the key must be provided in the code or manually if running interactively). A link to the relevant SAS documentation is in the references section.

General management of SAS data sets. PROC DATASETS is a powerful and versatile procedure. It can list the files in a library, delete files, copy files, change attributes, and perform many other tasks.

To list the files in a directory/libname:

```
proc datasets lib=####;
quit;
```

where #### above is replaced by the relevant libref.

To delete physical files:

```
proc datasets lib=####;
delete filename1 filename2;
quit;
```

To delete physical files and views in a single invocation:

```
proc datasets lib=####;
delete filename1 filename2 … / mtype=data;
delete viewname1 viewname2 … / mtype=view;
quit;
```

You will need the appropriate access permissions to delete files. The PROC statement options NOLIST and NOWARN are useful in some applications. The KILL option can delete **all** SAS files in a library, if ALTER= passwords are provided where needed. This option is very powerful and should be used with caution.

SAS catalogs. PROC CATALOG supports some of the same functionality – for SAS catalog files – that PROC DATASETS does for SAS data files. Catalogs, with the exception of large format libraries, usually require less space management than data set libraries.

SAS SQL file deletion. The DROP statement in PROC SQL & PROC FEDSQL can be used to delete files. There are 3 forms of the statement:

- DROP TABLE – for physical files
- DROP VIEW
- DROP INDEX – for data set indexes.

The DROP statement will delete AES encrypted files (without asking for the key when running interactively), unless they are also protected by an ALTER= password.

More advanced SAS tools for managing libraries: functions

SAS DATA step functions can be used for file management, if desired. These functions can be used to obtain a list of files in a directory, without the need to use external shell commands. The code to accomplish this can be found in Hamilton (2015) or the SAS macro language documentation (URL in reference section).

If the goal is to list all files in a directory and in all nested subdirectories, the problem is more challenging; Hamilton (2015) includes recursive code to accomplish this task. The functions described in the paper will let you search a directory and get the associated file names.

Once you have a list of files in a directory and confirmed that it contains files of the target type (usually .sas7bdat files), you can use the FDELETE function to delete a directory or a file in a directory. The assumption here is that you have the required operating system and/or metadata permissions to be able to delete files; the FOPTNAME function may be useful in some environments to check operating system permissions associated with a file.

These functions can be used to automate the file deletion process, which can be useful if there is a large number of files to be deleted. Alternately, a large number of files can be deleted using PROC DELETE or DATASETS, driven by user-written SAS macros.

TECHNIQUES THAT PROMOTE EFFICIENT USE OF DISK SPACE: FILE COMPRESSION AND USE OF FORMATS

The simple step of compressing a SAS data set can dramatically reduce the disk space required for a file. The SAS system supports data set compression in a number of ways:

- As a system option: OPTIONS COMPRESS=YES|BINARY|CHAR|NO and compression is applied to all files written in a program
- The same feature is available as a per-file data set option: (COMPRESS=YES|BINARY|CHAR|NO)

OPTIONS COMPRESS=YES; may be your system default. To check if it is the default, run this code **before** running any other code, right after login:

```
proc options;
run;
```

which gives a report with the value of all system options at run time.

System option: COMPRESS=YES|CHAR is most effective for data sets that are predominantly character variables. Predominantly character means that, when the data are divided into 2 categories, character vs. numeric variables, the space used per record (uncompressed) for character variables is significant. It does not have to be >50%; try compressed and uncompressed and see whichever uses the least space.

System option: COMPRESS=BINARY is most effective for files that are predominantly numeric variables (by space per record) AND have a large number of repeated values – like missing, 0,1,2, etc.

If the system option COMPRESS=YES is set on startup, you can make judicious use of the (COMPRESS=*) data set options to change the default compression option when appropriate. You can also reset the system option to COMPRESS=NO, but this should be done only when justified, e.g., the program creates very small files that when compressed are larger than when uncompressed. The data set options (COMPRESS=*) can also be used to override default settings on a by-dataset basis.

Data set option REUSE=YES. This option tells SAS to reuse any disk space freed by logical deletions, in compressed files only. This option does not impact uncompressed files. This option is potentially useful if your application is doing logical row deletions (additional discussion below).

PROC FORMAT: user-managed compression for select variables. If a large dataset has numerous character variables that are long in length and these variables have a limited number of values, they can be recoded to shorter strings via a custom format created with PROC FORMAT. For example, the format would recode:

“Very long string …. #1” = “Short 1”
“Very long string …. #2” = “Short 2”

and similarly up to string #n, with the short-form of the variable stored in the resultant file. To restore the variables back to their long-form, another format – the reverse of the above – is needed.

The advantages of this approach are as follows:

- if the short strings are meaningful, they can be a good substitute for the long strings;
- short strings are much easier to work with in code;
- can save a large amount of space in files with numerous long character variables.

The disadvantages:

- short strings may be cryptic for some long text variables;
- Need to know about the SAS format and apply it to get full text fields. This can be mitigated by providing views with the full-text, but such views require the user to specify LIBNAMEs to the source file, format library, and FMTSEARCH= OPTIONS.
- 2 formats to create and maintain. Reruns/fixes are usually required when there are unexpected changes in the long character variables,

Despite the limitations, in contexts where this approach is feasible, it can reduce disk space requirements.

LOGICAL VS. PHYSICAL DELETION OF RECORDS: LOGICAL ROW DELETION CAN PRODUCE HUGE FILES AND WASTE SPACE

SAS supports by-row logical deletion of records in SAS data sets:

- PROC SQL, PROC FEDSQL: DELETE statement
- DATA step: DELETE statement when used with MODIFY (but not SET, MERGE, UPDATE)

Over time, extensive logical deletion of rows in a table can produce a huge physical file with far more deleted rows than active rows (i.e., fewer logical rows than physical rows). Such files waste disk space and take much longer to read. Large reductions in file-read time and the release of large amounts of unused disk space may be achieved by rewriting the file via a DATA step, PROC COPY, or other means. Caution must be exercised when creating a new version of the file, as you need to preserve – if present:

- Sort order
- Indexes
- Integrity constraints, including referential integrity constraints.

One approach to regenerating data sets is to:

- Use PROC CONTENTS (or PROC DATASETS with CONTENTS statement) with OUT2= option to get the integrity constraints (if any)
- Make a new copy of the file using the DATA step (the copy has a different name), ignoring indexes, constraints
- Re-SORT the copy if needed
- Delete or rename old file
- Rename new file to target name
- Use PROC DATASETS to restore integrity constraints and/or indexes with constraint statements previously obtained via PROC CONTENTS.

PROC SQL can be used instead of PROC CONTENTS/DATASETS to get the integrity constraints for a SAS data set.

Identifying files that waste disk space. Appendix 2 contains prototype SAS code that uses the dictionary tables to identify SAS data sets that have a mismatch, number of physical vs. logical records. That code can be used to identify data sets that are wasting disk space, for eventual corrective action. (Also see SAS Usage note 32042; URL in references section.)

SAS PROGRAMMING TECHNIQUES TO REDUCE USE OF DISK SPACE

SAS DATA step vs. SQL. Depending on the processing being done and context/circumstances:

- Multiple SQL steps may be replaced with a smaller number of DATA/SORT/APPEND steps
- Alternately, multiple DATA/SORT/APPEND steps might be replaced with fewer SQL steps.

PROC DS2 is an object-oriented language that shares some commonality with the SAS DATA step. There are applications where DS2 is more efficient and might replace multiple DATA steps, and the opposite is true: there are applications where the DATA step is optimal. Matching the tool used per the requirements vs. application can promote efficient programming and reduce the use of disk space. If your site has the relevant SAS software products, DS2 processing can be exported and done in external databases. Exporting your processing to a relational database can reduce use of disk space and also reduce CPU usage on your local SAS server.

Useful data set options and related statements. Certain data set options and similar DATA step statements can be used to reduce I/O by limiting processing to only the variables/rows of interest. These data set options are:

1. KEEP=
2. DROP=
3. RENAME=
4. WHERE=.

#1-3 above operate on columns/variables, while #4 operates on rows.

There are similar DATA step statements: DROP, KEEP, RENAME, WHERE. The data set options function at the PDV: *program data vector level,* and the processing occurs when a data set is input or output. They are usually more efficient in use of disk space and CPU/memory, compared to the similar DATA step statements.

The above options can also be used with data sets in PROC SQL. You should be cautious about using SELECT * in SQL for large tables, unless there is good reason for its use.

Programming style. It is not uncommon to see programs written in SAS that are a number of small DATA steps, PROC SORT invocations, and maybe PROC SUMMARY or PROC MEANS. Review of the code inside the DATA steps often reveals that very little actual processing is being done. One DATA step with minimal processing is followed by another, similar DATA step with limited processing.

Code like this can usually be rewritten to accomplish the same functionality while using fewer DATA steps – possibly by using VIEWs instead of DATA steps (in some instances), and/or the functionality can be combined in a few PROC SQL invocations. This is something to look for in code reviews.

Program architecture: when possible, lengthy, complex programs should be divided into multiple (smaller) logical layers with related processing. Such programs may produce a derivative file that is the end product of, say n>1 steps, yielding (n-1)>0 intermediate files.

If these intermediate files occupy substantial disk space, PROC DATASETS and/or PROC DELETE can be used inside the logical layers to delete – as the processing progresses – intermediate files that are no longer needed. If the intermediate files may serve a debug function if a process fails, the deletion can be done under macro control. (This is very useful when writing large files in the WORK library.)

Using VIEWs, macros, and stored processes to reduce disk space usage. Intermediate and advanced SAS programmers should know how to use DATA step and SQL views. A VIEW is a compiled DATA step (or SQL) program that creates an interim and temporary physical file when the view is invoked/run. The motivation for using views is that they allow you to keep 1 copy of physical files, then

combine, extract, and filter to produce an interim derivative file as needed. Once the interim file has been used, it is automatically deleted by the SAS system (reducing overall disk usage). The view contains the instructions to create the target file, and usually occupies very little disk space.

Macros can be used in any SAS environment to run code and dynamically join files and/or filter the results. Stored processes are programs that run in the client-server environment and can be interactive (if desired). With suitable coding, in some contexts stored processes can be used to dynamically create desired data sets and avoid redundant disk storage.

Consider a more normalized data structure. Instead of large, denormalized files with hundreds of variables, consider a more normalized data structure that keeps only 1 copy of important parameters, and views are used to dynamically combine the data for reporting and analytics. Similarly, for sparse data, a variable that has – for example – 200 non-missing values in 600K rows – should be stored in a table of only 200 rows rather than 600K rows.

Reduce the lengths of character and numeric variables? A review of (legacy) SAS User Group papers finds multiple suggestions to reduce the length of stored numeric variables and character variables in SAS data sets. Keeping character variables at minimum length means the length may have to be increased later, or, if this task is forgotten, downstream reports and files may break later when the variable is used to store new, too-long values. Reducing the length of numeric variables may cause loss of precision.

File compression will usually release the blank space at the end of long character variables, so there is little to gain from reducing the length of character variables. Reducing variable length to the absolute minimum possible is not a good idea and is not recommended.

MANAGING SAS WORK LIBRARIES

If your program is processing small or medium-sized data files & data volumes via the SAS WORK library, then you probably don't need to be concerned with the disk space used by the WORK library. If your programs use large amounts of WORK library space, be aware that:

- The disk space for WORK libraries is shared by **all** SAS processes on your server and is **finite**,
- The WORK library disk space used by a process is cleared when the process terminates normally. If the process terminates abnormally or hangs, the disk space might not be cleared.
- Processes that use large amounts of disk space and hang can cause problems for other users, by effectively reducing the size of the WORK library. If your project/job/program hangs when it is running, the disk space is **not** cleared until a cleanup program runs – or – until you access the operating system shell and kill the process, or ask your Admin to do the same.

All of the suggestions above for making efficient use of disk space can be used in/with the WORK library.

Most of the time programmers don't need to know the physical location/address of their work library. WORK libraries are assigned dynamically and each SAS session has a unique directory address for its WORK library. If in fact you need to know the location (path) to the WORK library for your session, it can be found via the code:

```
%let address = %sysfunc(pathname(WORK));
%put &address;
```

You might want to include the code above in your autoexec process so the information is available in the log for each session.

You can redirect files that would be written in the default WORK library location, to an alternate location instead via the USER= system option. This code:

```
libname test "pathname";
options user=test;
```

will redirect any/all files from the default WORK library to the location of the libref/LIBNAME **test**, until a different USER= statement is entered or the program ends. This option is useful for programs that need very large amounts of WORK space.

At the time this is written (May 2017), the USER= option is not supported by the in-memory Cloud Analytic Services (CAS) component of SAS Viya. Billings & Alwani (2015) describes an alternate approach - a macro-variable-based method that accomplishes the same functionality as USER= but with greater flexibility, and it works in both Base SAS and CAS.

SUMMARY

- Basic housekeeping:
  - Delete redundant files
  - Move files that are no longer in active use but need to be saved, to archive/backup storage
- SAS procedures: DELETE, DATASETS, CATALOG can be used to delete files and manage libraries
- Zip, tar, and Git repositories can be used for data files, programs stored in archives
- Select SAS DATA step functions can support file management (advanced users)
- SAS supports file compression - character-based and numeric based (CHAR, BINARY). Can be set at system or at data set option level. This can save significant disk space and is suggested as a standard practice.
- PROC FORMAT supports user-managed recoding of long strings to short strings (also the reverse), and in some contexts this can save disk space.
- Logical deletion of rows can waste space in physical files; monitor the files and re-create the file to free up disk space, as needed.
- Programming techniques can make a big difference in efficiency/disk space usage:
  - SQL vs DATA/SORT/DS2 steps
  - Combine multiple small steps into fewer DATA, SQL steps
  - Advanced users: views, macros, stored processes
- Managing SAS WORK libraries:
  - USER= system option
  - More flexible macro-based alternative to USER=.

APPENDIX 1:
BSD 2-CLAUSE COPYRIGHT LICENSE (OPEN SOURCE)

```
* All program code in this paper is released under a Berkeley Systems
  Distribution BSD-2-Clause license, an open-source license that permits
  free reuse and republication under conditions;

/*
Copyright (c) 2017, MUFG Union Bank, N.A.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
```

```
2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
*/
```

APPENDIX 2: PROTOTYPE CODE TO IDENTIFY SAS DATA SETS WITH UNUSED DISK SPACE

The code below creates test data sets in the WORK library with artificial data and then deletes ½ of the records using SQL to create the type of file we want to detect. Librefs for the target directories are in a file accessed via an SQL subquery. If you are testing only 1 libref/directory then the subquery can be replaced with simpler code. The code below can easily be modified and encapsulated in a macro if desired.

Finally, the code identifies data sets with unused disk space but does not take any corrective action – that is left to the user. It also pulls metadata from the dictionary tables that specify compression and possible use of the REUSE= option. A comprehensive approach to disk space management would encourage file compression and, where rows are logically deleted, specifying the option REUSE=YES.

```
options nocenter dtreset;
%let nrows=1000000;

data control;
      length extra $4.;

      do i=1 to &nrows.;
            rnum = i;
            y = rand('uniform');
            extra = 'abcd';
            output;
      end;

      drop i;
run;

proc contents data=control;
      title "Contents before logical deletion";
run;

data log_del;
      set control;
run;
```

```
proc sql;
      delete from log_del
            where rnum > (&nrows./2);
quit;

proc contents data=log_del;
      title "Contents after logical deletion";
run;

data libref;
      length libname $8.;
      libname = "WORK";
      output;
      stop;
run;

PROC SQL;
   CREATE TABLE WORK.check_base AS
   SELECT a.libname,
          a.memname,
          a.memtype,
          a.modate,
          a.nobs,
          a.obslen,
          a.nlobs,
          a.nvar,
          a.compress,
          a.reuse,
          a.pcompress
      FROM dictionary.tables a
      WHERE a.libname IN (select libname from work.libref) AND a.memtype =
'DATA'
      ORDER BY a.libname,
               a.memname;
QUIT;

data work.flagged_files;
      set work.check_base;
      by libname memname;
      length unused_space $1.;
      unused_space ="N";
      unused_space_pct = .;
      est_file_size = .;
      length defect $1.;
      defect = "N";

      if (missing(nobs)) or (nobs = 0) or (nvar = 0) then
            do;
                  defect="Y";
                  return;
            end;

      est_file_size = nobs*obslen;

      if (nobs = nlobs) then
            return;
      else
```

```
            do;
                  unused_space="Y";
                  unused_space_pct = ((nobs-nlobs)/nobs)*100.;
            end;

      drop memtype;
run;
```

REFERENCES

Note: all URLs quoted or cited herein were accessed in September 2016.

Billings, T; Alwani, A (2015). Tips for Managing SAS Work Libraries. *SAS Global Forum 2015 Conference*. URL: https://support.sas.com/resources/papers/.../3196-2015.pdf

Hamilton J. (2012). Obtaining A List of Files In A Directory Using SAS® Functions. *Western Users of SAS Software Conference Proceedings.* URL: http://www.wuss.org/proceedings12/55.pdf

SAS Institute, Inc. (2016) online documentation:

- Cross-Environment Data Access (CEDA). *Moving and Accessing SAS(R) 9.4 Files, Second Edition.* URL: http://support.sas.com/documentation/cdl/en/movefile/67439/HTML/default/viewer.htm#p0c0l7xkprukh1n1ey1voicmgn6f.htm
- DATASETS Procedure. *Base SAS(R) 9.4 Procedures Guide, Sixth Edition*. URL: http://support.sas.com/documentation/cdl/en/proc/68954/HTML/default/viewer.htm#p16vqq5oedlmkin1373cqyalhpo6.htm
- DELETE Procedure. *Base SAS(R) 9.4 Procedures Guide, Sixth Edition*. URL: https://support.sas.com/documentation/cdl/en/proc/68954/HTML/default/viewer.htm#n0u6bw4nz2vrvtn1xvnqeyf7l7wn.htm
- Example 2: List All Files within a Directory Including Subdirectories. *SAS(R) 9.4 Macro Language: Reference, Fourth Edition.* URL: https://support.sas.com/documentation/cdl/en/mcrolref/67912/HTML/default/viewer.htm#n0js70lrkxo6uvn1fl4a5aafnlgt.htm
- FDELETE Function. *SAS® 9.4 Functions and CALL Routines: Reference, Fifth Edition*. URL: https://support.sas.com/documentation/cdl/en/lefunctionsref/67960/HTML/default/viewer.htm#p0h945u5r0cv6yn1u6qs35hiqt9t.htm
- FOPTNAME Function. *SAS® 9.4 Functions and CALL Routines: Reference, Fifth Edition*. URL: https://support.sas.com/documentation/cdl/en/lefunctionsref/67960/HTML/default/viewer.htm#n1ocdzkn9uhsa7n1qv3wwp4d3dot.htm
- Usage Note 32042: Deleting observations and reclaiming disk space. URL: http://support.sas.com/kb/32/042.html

CONTACT INFORMATION

A list of the author's SAS-related papers, including URLs for free access, is available at the URL (hosted by Google Drive): https://goo.gl/uCUHoa

Your enterprise web filter might prevent access to this URL from work, in which case you will need to access via a personal device.

Thomas E. Billings
MUFG Union Bank, N.A.

Remote from:
Merritt Island, Florida 32952

Phone: 321-453-5694
Email: tebillings@gmail.com