

# Outta Space with SAS® Software?

by Philip R Holland, Consultant, Holland Numerics Ltd, UK

## Abstract

When running SAS® programs on a PC, the user needs only to worry about how much free space there is on the PC they are using. However, when SAS programs are run under MVS on a mainframe machine, the correct use of disk space for WORK and permanent SAS Data Libraries is more complex, but essential to their smooth running. This paper describes various methods of exploiting the space available on a mainframe.

## Introduction

To demonstrate the similarities and differences between PC SAS and MVS mainframe SAS Systems I will include a quick comparison of PC and MVS mainframe terms here:

PC	MVS mainframe
RAM, e.g. 64Mb (1 user)	Region, e.g. 8Mb (100's of users)
Hard drive, e.g. 3.2Gb (10's, or less)	DASD, or disk volume, e.g. 2.5Gb (100's, or more)
DAT, e.g. 4Gb (10's)	Tape, e.g. 2Gb (1000's)

Unlike other platforms for SAS software, when you create a file under MVS, you must specify the initial and maximum amounts of disk space you require, in advance. If this is not done correctly, then, either the file will run out of space prematurely, if you specify too small an amount, or the file will not be created at all, if you specify an initial size which is bigger than the free space on any of the available disk volumes.

## Space Allocation under MVS

Space allocations under MVS are made up of 2 distinct parts:

- Primary allocation determines how big the file will be when it is first created, and, hence, determines which disk volume it will be created on, i.e. one which has free space big enough to receive the new file. It should be noted that the initial space allocation can be satisfied using up to 4 separate pieces, called 'extents'. If more than 4 extents would be required to complete the Primary allocation, then the creation fails.
- Secondary allocations, or extents, are used when the file has additional data written into it, so that the original allocation is not big enough to hold all

the data. Up to a total of 16 extents, including the Primary allocation, can be created for a file containing a SAS Data Library, but the size of each additional extent may be reduced, if the biggest contiguous section of free space on the disk volume is smaller than the Secondary allocation size.

There are 4 basic units of space used when allocating a SAS Data Library under MVS, i.e. CYL (1 cylinder = approx. 0.75Mb), TRK (15 tracks = 1 cylinder), BLK (blocks = file blocksize), and bytes. In general SAS Data Libraries tend to be allocated in cylinders or blocks, e.g.:

```
OPTIONS BLKSIZE(DISK)=OPT;
LIBNAME libref 'project.group.type'
DISP=(NEW,CATLG) SPACE=(CYL,(10,5));
```

```
DATA libref.newfile;
SET lib.master;
.....data processing.....
RUN;
```

The above example shows the creation of a SAS Data Library called 'project.group.type' with a Primary allocation of 10 cylinders, and additional Secondary allocations of 5 cylinders each. As no record format information was specified, SAS software will create the Data Library in an optimal allocation, using a record format of FS (Fixed Spanned), with logical record length and blocksize suitable for the type of disk volume used, i.e. for 3390 devices these would both be 27648 bytes. In general, increasing the blocksize will reduce the disk space required to store the data, as there will be fewer blocks of data per megabyte of disk and hence fewer 'inter-block' gaps, which are used to separate and identify the individual blocks of data.

## Re-allocation of Space under MVS

Some implementations of System Storage Managers, such as HSM, can affect the Primary allocations of SAS Data Libraries. In particular, when a SAS Data Library is archived and restored, it is returned to a disk volume in a single Primary extent, which, if it previously had Secondary extents before the archiving, will be larger than originally specified. The Secondary allocation size will not be affected.

## Common MVS Disk Space Errors

Most MVS error codes related to having insufficient disk space to expand the SAS Data Library end in '37', e.g. SB37, SD37 and SE37. This is caused by not allocating

enough space for the amount of data being written to it. However, the amount of data actually being written may be greater than anticipated for several reasons:

- The code is looping around an OUTPUT statement.
- A PROC SQL join is generating every possible combination of each record in one SAS Dataset with records in another SAS Dataset.
- A piece of SAS code used to select specific records is not being executed, or is selecting every record.

## Saving Permanent Disk Space

Permanent disk space is used for storing SAS data to be retained after the SAS session has ended.

## Creating a SAS Dataset

There are many ways a SAS Dataset can be created, but, by default, the resulting Dataset will contain every column input to, and created by, the step processing. It is probable that some of these columns will not be used again in the program, so the space occupied by these columns is just being wasted. To save space just KEEP the columns you require in a Data step, SELECT the columns you require in PROC SQL, or DROP the columns you do not need later in a Data step, e.g.:

```
DATA lib.newfile
  (KEEP=needed1 needed3 needed5);
  SET lib.master;
  .....data processing.....
RUN;
```

or

```
DATA lib.newfile;
  SET lib.master;
  KEEP needed1 needed3 needed5;
  .....data processing.....
RUN;
```

or

```
PROC SQL;
  CREATE TABLE lib.newfile AS
  SELECT needed1, needed3, needed5
  FROM lib.master;
QUIT;
```

or

```
DATA lib.newfile
  (DROP=temp2 temp4 temp9);
  SET lib.master;
  .....data processing.....
RUN;
```

or

```
DATA lib.newfile;
  SET lib.master;
  DROP temp2 temp4 temp9;
  .....data processing.....
RUN;
```

## Replacing a SAS Dataset

If the output SAS Dataset from a processing step already exists, then, to preserve the old Dataset from premature destruction, the new Dataset is created in free space within the SAS Data Library. Only when the processing step has successfully finished creating the new Dataset, is the old Dataset deleted, and the space it occupied added to the free space. This means that whenever a Dataset is copied within a SAS Data Library, there must be sufficient free space available to store another copy of that Dataset, and the original data gets moved to a completely different area within the Data Library as a result of this processing.

## Appending to a SAS Dataset

If you do not wish to rewrite the SAS Dataset every time it needs new records adding to it, try appending the new records to the end of the Dataset using PROC APPEND, e.g.:

```
DATA newrecs;
  column1='ABC123';
  column2=42;
  OUTPUT;
  column1='ABD153';
  column2=47;
  OUTPUT;
RUN;

PROC APPEND BASE=lib.master
  DATA=newrecs;
RUN;
```

Note that PROC APPEND expects the BASE= and DATA= SAS Datasets to have exactly the same columns. The FORCE parameter should be used to allow a Dataset containing new records with differing columns to be appended to the master Dataset without generating errors.

## Inserting and Deleting Observations

PROC SQL can be used to INSERT or DELETE records within a SAS Dataset.

The INSERT SQL statement works in a similar way to PROC APPEND, by adding new records at the end of the current Dataset. Any columns not specified in the INSERT statement are filled with missing values when the new record is written to the SAS Dataset.

The DELETE SQL statement will mark individual records as deleted, but does not remove the data from the SAS Dataset. This means that, over time, the number of undeleted records may not change dramatically, but the

size of the SAS Dataset will continue to increase. The only way to clean out the records marked as deleted is to copy the Dataset to a new location, e.g.:

```
DATA lib.master;
  SET lib.master;
RUN;
```

or

```
PROC SQL;
  CREATE TABLE work.master AS
  SELECT * FROM lib.master;
  CREATE TABLE lib.master AS
  SELECT * FROM work.master;
QUIT;
```

or

```
PROC COPY IN=lib OUT=work;
  SELECT master;
RUN;
```

```
PROC COPY IN=work OUT=lib;
  SELECT master;
RUN;
```

## Creating Multiple SAS Datasets in the Same Data Step

The technique of creating multiple SAS Datasets in a single Data step is not really a space saving technique, but will reduce the amount of data read, as, for each record read, there can be many records written in several different Datasets. The only other method available would be to read the input Dataset individually for each output Dataset created. For example:

```
DATA lib.file1 lib.file2 lib.file3;
  SET lib.master;
  .....data processing.....
  IF type=1 THEN OUTPUT lib.file1;
  ELSE IF type=2 THEN
    OUTPUT lib.file2;
  ELSE OUTPUT lib.file3;
RUN;
```

## Compression

It is possible, using the COMPRESS=YES SAS System or Dataset Option, to dramatically reduce the disk space used by a SAS Dataset. Compression ratios of up to 50% can be achieved on SAS Datasets containing mostly numeric data and a high proportion of zeroes and missing values, which could increase to 80% or more for SAS Datasets containing mostly character fields having a large proportion of blanks. Note that a high proportion of non-zero and non-missing numeric values could result in the compression process increasing the size of the SAS Dataset. The SAS System Option is used to set the default method of saving SAS Datasets, and the SAS Dataset Option allows the user to specify which SAS Datasets are compressed.

Like all benefits, there is a price to pay, as the saving in disk space causes an increase in the CPU time required to read the data, as the compressed data has to be decompressed before it can be read.

## Saving Temporary Disk Space

Temporary disk space is used to store intermediate SAS data, but is returned to the pool of temporary disk space and the data lost, at the end of the SAS session.

## WORK Allocation

By default SAS software allocates a Work Data library with the library reference of WORK. However, it is possible to allocate many library references to temporary work volumes, thereby allowing the operating system to spread the Work Datasets across many volumes, e.g.:

```
LIBNAME work01 '&temp1' UNIT=sysda
  SPACE=(CYL,(100,100));
```

```
LIBNAME work02 '&temp2' UNIT=sysda
  SPACE=(CYL,(100,100));
```

```
LIBNAME work03 '&temp3' UNIT=sysda
  SPACE=(CYL,(100,100));
```

You can then write your temporary SAS Datasets to one or other of the Work Data Libraries, e.g.:

```
DATA work01.file1 work02.file2
  work03.file3;
  SET lib.master;
  IF type=1 THEN OUTPUT work01.file1;
  ELSE IF type=2 THEN
    OUTPUT work02.file2;
  ELSE OUTPUT work03.file3;
RUN;
```

## Sort Work Allocation

By default in batch jobs SAS software allocates 3 sort work files, called SORTWK01-SORTWK03, to provide an overflow working space for PROC SORT, and other SAS procedures that order the data. In interactive SAS there are no sort work files allocated by default. The SAS System Option SORTWKNO can be used to change this number to a maximum of 6, and spread the sort overflow, e.g.:

```
OPTIONS SORTWKNO=6;
```

The SAS System Option SORT is used to change minimum total size of Sort Work datasets and defaults to 0 interactively and 4 in batch. It is measured in the units specified by the SORTUNIT System Option, which defaults to CYLS, e.g.:

```
OPTIONS SORT=10;
```

However, if you find you need to change these SAS System Options in interactive SAS software, then it probably means that you ought to be executing this SAS program in batch anyway.

## Reusing WORK Dataset Names and other Housekeeping

How many times have you run a SAS program which uses WORK Datasets for storing intermediate data and run out of space in your WORK Data Library before the end?

The data stored in these WORK Datasets is probably only needed for a small section of the program, and then sits there unused for the remainder of the run wasting vast amounts of WORK space. If you were to reuse some of the names of these WORK Datasets later in the SAS program, then you will effectively recycle their disk space, allowing you to temporarily save more data, e.g.:

```
DATA file10;
SET file9;
.....data processing.....
RUN;
```

```
DATA file9;
SET file10;
.....more data processing.....
RUN;
```

Alternatively you could delete all your WORK Datasets after you have finished with them, either individually, e.g.:

```
PROC DATASETS LIB=work;
DELETE file9 file10;
RUN;
```

or all at once, e.g.:

```
PROC DATASETS LIB=work KILL;
RUN;
```

## General Disk Space Economies

The following information has been included to supplement the suggestions for saving permanent and temporary disk space, and, in some cases, should not be used to save disk space unless there would be additional benefits associated with the any changes made.

### Tape Format SAS Datasets

The typical SAS Data Library on disk is allocated so that the data can be accessed at random, and includes a directory inside to point to the positions of individual SAS Datasets and other SAS objects within the Data Library. This additional information increases the size of the Data Library and prevents it from becoming larger than the size of a single disk volume.

If a SAS Data Library is created on a tape, the TAPE engine is automatically used to create a special sequential-format SAS Data Library. This format can only be written or read sequentially, which restricts the user to accessing a single SAS Dataset from any individual sequential-format SAS Data Library. However, a sequential-format SAS Data Library can be written across more than one tape volume.

It is possible to create sequential-format SAS Data Libraries on disk, instead of tape, volumes by using the TAPE engine keyword on the LIBNAME statement when creating a new SAS Data Library, e.g.:

```
LIBNAME libref TAPE
'project.group.type'
SPACE=(CYL,(10,10))
DISP=(NEW,CATLG);
```

However, sequential-format SAS Data Libraries allocated on disk volumes occupy about 50% more disk space than their equivalent standard SAS Data Libraries, so they are only recommended for tape storage.

### Transport Format SAS Datasets

Standard SAS Datasets on different platforms are structured such that a binary copy from one platform to another is likely to create an unreadable file on the target platform. To alleviate this problem SAS provide 2 different file formats which can be used to transfer SAS Datasets from platform to platform.

### XPORT Engine Files

These files can be created using a LIBNAME statement with the XPORT engine parameter. It is recommended that a record format of FB, record length of 80 and blocksize of 8000 be used for the file. This format allows SAS to read and write data in the file by just using the library reference. While this format is very straightforward to use, the transportability results in an increase in size of up to 15%.

```
LIBNAME trans XPORT
'project.group.type'
SPACE=(CYL,(10,10))
DISP=(NEW,CATLG)
LRECL=80 RECFM=FB BLKSIZE=8000;
```

```
PROC COPY IN=lib.master OUT=trans;
RUN;
```

### PROC CPORT/CIMPORT Files

These files can only be created using the CPORT SAS procedure writing to a target files allocated using a FILENAME statement. Like the XPORT engine files, it is recommended that a record format of FB, record length of 80 and block size of 8000 be used for the file. This format can only be read by SAS software once it has been converted back to a standard SAS format using the CIMPORT SAS procedure. The files can be smaller than the original SAS Data Libraries, provided they do not contain a high proportion of non-zero and non-missing numeric values, by as much as 40%.

```
FILENAME cport 'project.group.type'
SPACE=(CYL,(10,10))
DISP=(NEW,CATLG)
LRECL=80 RECFM=FB BLKSIZE=8000;
```

```
PROC CPORT DATA=lib.master FILE=cport;
RUN;
```

## Releasing Unused Space in a SAS Data Library

Finally, if you want to save some space on a disk volume in advance of the Archiving System's nightly clean-up, then PROC RELEASE will allow you to release the unused space at the end of the SAS Data Library to an extent boundary, e.g.:

```
LIBNAME libref CLEAR;

FILENAME fileref 'project.group.type'
  DISP=OLD;

PROC RELEASE DDNAME=fileref EXTENTS;
RUN;
```

Note that it will not release free space from the middle of the Data Library. The simplest method of freeing all the unused disk space from within a SAS Data Library is to create a new SAS Data Library, copy all the SAS objects from the old Data Library to the new with PROC COPY, delete the old Data Library, rename the new Data Library to the name of the old one, and finally run PROC RELEASE to release any unused space at the end of the new Data Library. The SAS objects will be located at the beginning of the new SAS Data Library as they are copied over.

This saving can also be achieved when allocating a SAS Data Library using the RLSE option in the SPACE parameter, e.g.:

```
LIBNAME libref 'project.group.type'
  DISP=(NEW,CATLG)
  SPACE=(CYL,(10,10),RLSE);
```

## Recommendations

- Don't keep columns of data in SAS Datasets that you will not use again. Use KEEP= or DROP= options, or KEEP or DROP statements, to cut down unnecessary waste.
- Don't keep WORK Datasets that you will not use again. Reuse their names, or better still delete them altogether, when you have finished using them, so that you can reuse the space they occupied.
- Compress SAS Datasets containing high proportions of blanks, zeroes or missing values, if you are running out of disk space, to make significant savings. However, beware of corresponding increases in CPU time to process these Datasets.
- Plan your disk space usage in advance. MVS files cannot be increased in size by user actions once they have been allocated. Only the automated system utilities can reallocate SAS Data Libraries,

and do so by removing them from the disks and then recreating them in a single extent.

## References and further reading

- SAS Companion for the MVS Environment, Version 6, Second Edition
- SAS Language: Reference, Version 6, First Edition
- SAS Procedures Guide, Version 6, Third Edition
- SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition

*SAS is a registered trademark of  
SAS Institute Inc., Cary, NC,*

## Contact Details

The author is a consultant for Holland Numerics Ltd and can be contacted at:

Philip R Holland  
Holland Numerics Ltd  
94 Green Drift, Royston, Herts. SG8 5BT, UK  
e-mail: [phil.holland@bcs.org.uk](mailto:phil.holland@bcs.org.uk)  
web: [www.hollandnumerics.demon.co.uk](http://www.hollandnumerics.demon.co.uk)  
tel. (mobile): +44-(0)850-295556