INSIDE SAS® CATALOGS IN VERSION 6 OF THE SAS SYSTEM
Rebecca Perry, SAS Institute Inc., Cary, NC

INTRODUCTION

A SAS® catalog is a SAS file used to store a special class of data objects called SAS catalog entries. Catalogs contain a variety of entries essential to Version 6 of the SAS System. Some of the data objects maintained in catalogs are

- formats generated by the FORMAT procedure
- graphs produced by SAS/GRAPH® software
- function key settings
- letters produced by SAS/FSP® software
- default option settings.

This paper describes the internal architecture of a Version 6 SAS catalog and presents options for optimizing the performance of a SAS catalog.

PART I: SAS® CATALOGS

SAS catalogs reside in SAS data libraries and have the library member type CATALOG. Like other SAS files, SAS catalogs have two-level names, as in the following example, where the first-level name, libref, is a name assigned to the SAS data library to which the catalog belongs, and the second-level name, catalog, is the name of the catalog:

libref.catalog

Catalog Entries

Catalog entries are generally referred to by a two-level name, as in the following example, where the first level is the entry name and the second level is the entry type:

KEYS.help

KEYS is the entry name and HELP is the entry type.

Different types of information are stored in different types of catalog entries. Different entry types can be stored in the same SAS catalog. The following list describes a sampling of some Version 6 catalog entry types:

- CBT entries contain computer-based training lessons.
- HELP entries contain help information for applications developed with SAS/AF® software.
- GRSEG entries contain graphs created by SAS/GRAPH software.
- PROGRAM entries contain source programs written in Screen Control Language (SCL).
- KEYS entries assign commands to function keys for use with the SAS Display Manager System.

A Brief Comparison of Version 5 and Version 6 Catalogs

Version 5 of the SAS System supports two types of catalogs: full-screen catalogs (MEMTYPE=CAT) and graphics catalogs (MEMTYPE=GCAT). In Version 6, all catalogs have the same format and have the library member type CATALOG. Some things that were separate, loose data library members in Version 5, such as ETS models, formats (on VMS® Data General, and PRIMOS®), and SAS/IML® storage libraries, are stored in catalogs in Version 6.

In Version 6, the use of catalogs is more widespread throughout the SAS System than in Version 5. Consequently, more catalog entry types are supported in Version 6.

The VSTOV6 procedure converts Version 5 SAS files to Release 6.06 format. PROC VSTOV6 can convert an entire SAS data library, a specific catalog within a library, or selected entries within a catalog. For more information, refer to Helen Wolfson’s paper, “Converting Version 5 Files to Release 6.06 Files Using the VSTOV6 Procedure,” which will be published in the 1990 SAS Users Group International Conference Proceedings: SUGI 15.

SAS catalogs have better integrity protection in Version 6 than in Version 5. If a Version 6 catalog’s integrity is questionable after a disk-full condition or an I/O error, the SAS System flags the catalog as damaged. A facility for repairing the damaged catalog is available.

Layering of the SAS® System

SAS software is composed of three main layers. The applications or procedure code is the top layer, the supervisor/engine code is the middle layer, and the host code is the bottom layer. The host layer is machine- and operating-system-dependent. This paper focuses primarily on the processing of catalogs within the supervisor layer.

Architecture of a Version 6 Catalog

Catalogs are made up of a variety of components. Each component is stored in logical units called pages. The host layer of code maps pages into physical units on disk called blocks. On some operating systems, a page maps directly into a block on disk; on others, a page can be divided into smaller blocks or combined into one large block. Thus, the physical transfer of data between disk and memory during a read or write operation can be a page, a section of a page, or a block of pages. The unit of transfer between the supervisor layer and the host layer is always a page.

The host layer of code determines an optimal page size for a catalog based on the physical characteristics of the mass storage on which the catalog will reside. On some hosts, such as MVS, you can change the default block size associated with a particular device type by specifying the BLKSIZE= option. The default block size associated with a device type determines both the default SAS library block size and the page size for catalogs stored in that SAS library.


A catalog is made up of five components:

- the catalog directory
- the catalog header record
- the subdirectory records (one per catalog entry)
The catalog directory contains the names of the entries in the catalog, along with a 1-byte flag and the address of a subdirectory record for each entry. The subdirectory record contains more information that describes the entry. The flag indicates whether the entry should be displayed in catalog directory listings or remain hidden from users. The entry name portion of each directory record is called the key and is stored as entry type name followed by entry name.

Figure 1 represents the format of the directory record for catalog entry KEYS.HELP.

```
HELP KEYS   flag   addr of KEYS.HELP subdirectory record
```

Figure 1 Directory Record for KEYS.HELP

The catalog's directory is stored in the form of a tree structure, as shown in Figure 2. The tree has a single root, branches, and leaves at the end of the branches. By convention, the root of the tree is the highest level in the tree, and the leaves are at the lowest level (level 0).

![](image)

Figure 2 Catalog Directory Tree Structure

The directory records for each catalog entry are stored in the leaf nodes of the tree. They are also stored in the directory sorted by the entry's key (entry name within entry type name). The lowest key in the example directory tree is CBT A, so the directory record for this key is stored in the left-most leaf node.

Each nonleaf node of the tree contains pointers to the nodes on the level below it (children nodes) and the value of the highest key that can be stored in each child node. All nodes at the same level are linked together by right pointers.

The search for an entry name or key in the directory begins at the root node and moves down through one node at each level until the leaf node that should contain the key is found. To locate the key PROGRAM A in Figure 2, the tree is traversed as follows:

1. Visit the root node. PROGRAM A is less than PROGRAM MY, so you follow the left tree branch.
2. Visit the level 1 node containing the high key values CBT Z, HELP ME, and PROGRAM MY. PROGRAM A is greater than HELP ME but less than PROGRAM MY, so you follow the right-most branch of the tree.
3. Visit the leaf node. PROGRAM A is the second entry in the node.

If the tree is traversed to a leaf node and the key searched for is not in the leaf node, then the key is not in the catalog. Once the key is located, keys higher than that key are found by reading the remaining keys in the same node, then following the right pointer to nodes containing higher keys.

A tree is balanced when all leaves are the same distance from the root. This attribute of a tree is important because it provides a uniform cost to access any leaf node. The SAS catalog directory is automatically kept balanced as updates are made. This is done by making slight changes to the distribution of keys across nodes at the level where imbalance is detected.

Space occupied by deleted directory records is recovered and reused when new directory records are added to the catalog's directory.

The catalog's directory tree is stored in pages on disk. Each node of the directory tree maps to one page on disk. Pages are added to the directory as the number of entries in the catalog grows. These new directory pages are interspersed with other pages of the catalog.


### The Catalog Header Record

The second catalog component, the catalog header record, contains information about the catalog as a whole. Some components of the catalog header record are:

- catalog page size
- count of the number of entries in the catalog
- information used to manage free space within the catalog
- information used to manage the catalog's subdirectory pages
- damage flags.

### The Catalog Subdirectory Records

The subdirectory records are the third component of a catalog. Each entry in the catalog has a subdirectory record, which contains the following information about the catalog entry:

- date and time the entry was last updated
- 40-byte description of the entry
- size of the data blocks for the entry
- address of the first data block for the entry
- count of the number of data blocks for the entry

868
The pages of the catalog. The first block of subdirectory records and the catalog header record map onto the same catalog page. Subdirectory pages are added to the catalog as they are needed, and the added subdirectory pages are interspersed with other pages of the catalog.

Catalog Entry Data Blocks
The fourth catalog component comprises data blocks that contain the data stored in a catalog entry. For each catalog entry, there are one or more data blocks. The data blocks for an entry are linked together by forward pointers (see Figure 3).

All data blocks for an entry will be the same size, but different entry types may have different data block sizes. Six different data block sizes are available. The choices for block size are host-dependent and range from 64 bytes to 32K bytes. Generally, the SAS System selects a block size that is optimal for a particular entry type, based on the record length and number of records that entry type usually contains. Many entries have a data block size equal to the catalog page size, especially when the amount of data that a user will store in that entry type is not predictable.

Data blocks for an entry are mapped onto pages of the catalog. If an entry requires multiple data blocks, the blocks will usually be contiguous on the same page. Data blocks for an entry can span multiple pages. These pages are usually, but not always, contiguous.

In Figure 3, the data blocks for entry Y span across 2 pages. The first two data blocks for entry Y are on the first page, while the third data block for entry Y is at the beginning of the second page. The mapping of data blocks onto catalog pages affects I/O performance and is discussed in Part II of this paper.

Each data block has 16-byte data block header record at the beginning.

The block header record contains
- the page number that particular data block maps onto
- the block size of that data block

- a pointer to the next data block for that entry or an end-of-data marker if this is the last data block for that entry.

Catalog Entry Data Record Format
Records that contain the entry's data follow the data block header. Each record is preceded by a 4-byte record header containing the record type and record length for that record. Data records can vary in length from 1 to 32K bytes. Figure 4 shows an example of the format of an entry's data record.

Figure 4 Format of an Entry's Data Record

Records are stored in data blocks in the order the SAS System writes them. As many records as possible are packed into a data block. If a complete data record will not fit into the remaining space in a data block, a new data block is added to the entry. The remaining portion of the record is written to the next data block. The newly added data block is linked to the previous data block by storing its address in the block header of the previous data block.

The data records for an entry can be compressed. The SAS System automatically compresses many types of catalog entries. Some types of entries contain data that do not benefit much from compression. The data for these types of entries are left uncompressed.

The compression algorithm used for catalogs is simple: any occurrence of 3 to 126 consecutive, identical characters within the same record compresses into 3 bytes (if the compression option is enabled for the entry). The compressed sequence consists of
- the escape character (01)
- a count of the number of characters compressed
- the character itself.

For example, the data string SSSSSSSSS compresses into the sequence 185.

User-supplied compression algorithms for catalogs are being considered for a future release. Such a function could be designed to compress specific classes of data better than the SAS System general purpose compression algorithm.

The Bitmap Entry
The last catalog component we will examine is the internal catalog entry called the bitmap entry. The bitmap entry, or bitmap, contains information used in managing space within the catalog's data block pages. Each data block page in the catalog maps to a 4-bit area in the bitmap. Bits in the 4-bit area indicate whether any free blocks are available on that page, and if free blocks are present, what size they are.

I will explain later how the bitmap is used when adding data blocks to the catalog. Management of free space within the catalog has a direct effect on performance.

Examples of Catalog Processing
Let's look at how catalogs are processed using a specific SAS System example. How are catalogs used when browsing a catalog LETTER entry using the FSLETTER procedure? And what catalog processing takes place when the directory of a catalog is displayed?
Suppose you have a catalog named MYLIB.LETTERS that contains 300 entries. If you submit the following SAS statements, the FSLETTER DIRECTORY window is displayed.

```sas
proc fsletter c=mylib.letters;
run;
```

The entry name, entry type, entry description, and date the entry was last updated will be listed for all 300 catalog entries. A partial listing of these catalog entries is shown in Output 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>L. A100</td>
<td>LETTER</td>
<td>Manual Letter</td>
<td>03/01/88</td>
</tr>
<tr>
<td>L. A101</td>
<td>LETTER</td>
<td>Screening Letter</td>
<td>03/02/89</td>
</tr>
<tr>
<td>L. A102</td>
<td>LETTER</td>
<td>Allowing Letter</td>
<td>03/03/90</td>
</tr>
<tr>
<td>L. A103</td>
<td>LETTER</td>
<td>Elementary Letter</td>
<td>03/04/91</td>
</tr>
<tr>
<td>L. A104</td>
<td>LETTER</td>
<td>Manual Letter</td>
<td>03/05/92</td>
</tr>
<tr>
<td>L. A105</td>
<td>LETTER</td>
<td>Example Letter</td>
<td>03/06/93</td>
</tr>
<tr>
<td>L. A106</td>
<td>LETTER</td>
<td>Example Letter</td>
<td>03/07/94</td>
</tr>
<tr>
<td>L. A107</td>
<td>LETTER</td>
<td>Licensing Letter</td>
<td>03/08/95</td>
</tr>
<tr>
<td>L. A108</td>
<td>LETTER</td>
<td>ND Association Letter</td>
<td>03/09/96</td>
</tr>
<tr>
<td>L. A109</td>
<td>LETTER</td>
<td>PDA User</td>
<td>03/10/97</td>
</tr>
<tr>
<td>L. A110</td>
<td>LETTER</td>
<td>Move Letter</td>
<td>03/11/98</td>
</tr>
<tr>
<td>L. A111</td>
<td>LETTER</td>
<td>Example Letter</td>
<td>03/12/99</td>
</tr>
<tr>
<td>L. A112</td>
<td>LETTER</td>
<td>Manual Letter</td>
<td>03/13/00</td>
</tr>
<tr>
<td>L. A113</td>
<td>LETTER</td>
<td>Example Letter</td>
<td>03/14/01</td>
</tr>
<tr>
<td>L. A114</td>
<td>LETTER</td>
<td>Manual Letter</td>
<td>03/15/02</td>
</tr>
<tr>
<td>L. A115</td>
<td>LETTER</td>
<td>Example Letter</td>
<td>03/16/03</td>
</tr>
<tr>
<td>L. A116</td>
<td>LETTER</td>
<td>Licensing Letter</td>
<td>03/17/04</td>
</tr>
<tr>
<td>L. A117</td>
<td>LETTER</td>
<td>Letter's Registration Notice</td>
<td>03/18/05</td>
</tr>
<tr>
<td>L. A118</td>
<td>LETTER</td>
<td>Screening Letter</td>
<td>03/19/06</td>
</tr>
<tr>
<td>L. A119</td>
<td>LETTER</td>
<td>Screening Letter</td>
<td>03/20/07</td>
</tr>
<tr>
<td>L. A120</td>
<td>LETTER</td>
<td>Screening Letter</td>
<td>03/21/08</td>
</tr>
</tbody>
</table>

Output 1 FSLETTER Directory Window

Catalog Directory Window Displayed by the FSLETTER Procedure

Let's look behind the scenes at how the previous operations affect memory and I/O requirements in the supervisor layer. This paper does not discuss how these operations are handled at the host layer because each Version 6 host layer has its own scheme for buffering and doing actual I/O.

In the example, the catalog MYLIB.LETTERS has a page size of 6144 bytes. The page size has a direct effect on the number of page requests the portable layer makes to the host layer during catalog processing.

With a page size of 6144, 46 subdirectory records will fit on a subdirectory page. The catalog requires six subdirectory pages. A maximum of 290 keys will fit onto a directory page of size 6144. The catalog MYLIB.LETTERS has 300 entries, yet only 290 keys will fit onto a directory page, so a two-level directory index is required. The directory for MYLIB.LETTERS requires three pages: the root page and two children pages. Remember that the root page contains no actual key entries. The root page of the directory of MYLIB.LETTERS contains the highest key that will be allowed in each of its child pages. In the example catalog, the first 150 keys will be stored in the left-most child page, and the remaining 150 keys will be stored in the right-most child page, as shown in Figure 5.

When you submit the following statements, certain catalog processes occur in order to display the catalog directory screen.

```sas
proc fsletter c=mylib.letters;
run;
```

These catalog processes take place in the following order:

1. The catalog is opened in UPDATE mode.
2. The catalog’s directory is opened.
3. The catalog is positioned at the beginning of the directory.
4. Information about each catalog entry is read from the catalog and displayed. For each entry, the system must
   - Locate the entry’s key in the directory (part of the key contains the subdirectory record address).
   - Read the subdirectory record for the key just fetched (to get the entry description and the data of last update).
   - Write directory information for this entry to the FSLETTER DIRECTORY window.
5. The catalog’s directory is closed.

Let’s look at how these processes affect I/O requirements in the supervisor layer. When the example catalog is opened, the supervisor layer requests the following four pages:

- the directory root page
- the catalog header page (which also contains subdirectory records for first 28 catalog entries)
- the page of the directory that contains the bitmap entry’s key (the right-most child page of the directory)
- the page containing data for the bitmap entry.

Opening the catalog’s directory generates another page request: the left-most child page of the directory that contains the lowest key in the directory (LETTER A100).

Reading through all keys in the directory and fetching the subdirectory record for each key read generates six more page requests. All six page requests are for subdirectory pages.

Now let’s see how these catalog processes affect memory requirements in the supervisor layer.

In the example earlier, opening the catalog MYLIB.LETTERS in update mode causes six page size buffers to be allocated. The buffers hold the catalog’s data while it is being processed. The number of buffers allocated to open a catalog is proportional to the number of levels in the directory tree. Additional buffers allocated for multilevel directories will hold directory pages.

Opening the directory of MYLIB.LETTERS causes two more page size memory buffers to be allocated. The number of buffers allocated by a directory open is proportional to the number of keys in the catalog. The extra buffers allocated for catalogs with many keys will hold subdirectory pages.

Closing the directory of MYLIB.LETTERS frees the two buffers allocated during the directory open.
Suppose you want to browse the letter entry A150.LETTER. If you tab down to the A150.LETTER entry and enter a B (for BROWSE), another window appears, displaying the contents of the A150.LETTER entry, as shown in Output 2.

Output 2  FSLETTER BROWSE Window

Contents of A150.LETTER Displayed by the FSLETTER Procedure

Browsing the contents of A150.LETTER causes the following catalog processes to occur:

1. The entry A150.LETTER is opened for READ access.
   - The key LETTER A150 is located in the directory (part of the key contains the subdirectory record address).
   - The subdirectory record for entry A150.LETTER is read (to get the address of the first data block for this entry).

2. All records for the entry A150.LETTER are read.

When entry A150.LETTER is opened, the supervisor layer requests the following pages:

- the subdirectory page that contains subdirectory record for A150.LETTER
- the page containing the first block of data for entry A150.LETTER

Note that no I/O requests were generated for the directory pages of the catalog. All three directory pages still reside in the buffers into which they were read during earlier catalog open and catalog directory open operations.

Reading the records for the entry A150.LETTER generates no additional I/O requests. All of the data records for entry A150.LETTER are contained on the page that was read when the entry was opened.

Opening the entry A150.LETTER allocates one memory buffer the size of the entry's block size (6144 in the example). This buffer will hold the current block of data for A150.LETTER as the entry's data records are read.

Free Space Management Within a Catalog

Editing a catalog entry or creating a new catalog entry may require the addition of new data records for the entry. Let's look at the methods used to acquire space in the catalog for new records. These methods involve attempting to locate free space available in the catalog and extending the file size of the catalog when necessary.

When creating a new catalog entry, at least one data block must be acquired for the new entry. The following steps are taken:

1. Attempt to get the data block from the last page of the file. If the last page does not have a large enough free block, then go to step 2.
2. Search the bitmap for a page that contains a block that is equal to or greater than the required block size. If no pages represented in the bitmap contain a large enough block, then go to step 3.
3. Extend the file by a page and allocate the requested block from this page.

When adding data records to an existing catalog entry, the following steps are taken:

1. Attempt to write the new records after the last data record in the entry's last data block. If there is not enough room in the last data block and a new data block must be acquired then, go to step 2.
2. Search the bitmap for a page that contains a block that is equal to or greater than the required block size. Start searching the bitmap at the location that represents the page that contains the last data block for this entry. If no pages represented in the bitmap contain a large enough block, then go to step 3.
3. Attempt to get the data block from the last page of the file. (The last page of the file is not represented in the bitmap). If the last page does not have a large enough free block, then go to step 4.
4. Extend the file by a page and allocate the requested block from this page.

Note that if several entries are being updated concurrently and new data blocks are being acquired for these entries, the data blocks for these entries can be interspersed, fragmenting the data for these entries.

When an entry is deleted, the data blocks for the entry are made available for reuse. This is done by:

- recombining data blocks that are adjacent to one another into larger blocks when possible
- updating the associated bitmap entry (for every page in which data blocks were freed) to reflect the largest block size now available on the page.

The catalog's space management algorithms attempt to keep data blocks for the same entry on the same page or pages that are adjacent to one another. But, inevitably, when a lot of update activity is done against a catalog, the catalog becomes fragmented. In Part II of this paper, you will see how to determine if a catalog is fragmented and how to correct the problem.

PART II : CATALOG PROCESSING ISSUES

Part II of this paper discusses detection and repair of damaged catalogs and presents options for optimizing the performance of a SAS catalog.
Damaged Catalogs

In Version 6 of the SAS System, if a catalog's integrity is questionable, the catalog is flagged as being damaged. Some situations that can cause a catalog to be marked damaged include the following:

- A system failure occurs while updating a catalog.
- The disk where the SAS catalog is stored becomes full before the catalog is completely written to disk.
- An I/O error occurs while writing to the catalog.

When one of these situations occurs, the entire catalog may be marked as damaged, or only specific entries within the catalog may be marked as damaged. Usually, only the entries open for update at the time of a disk-full or I/O error are marked as potentially damaged. But if an error situation occurs while writing directory pages or subdirectory pages to the catalog, the entire catalog is flagged as damaged.

When using the SAS System interactively, the next attempt to access a damaged catalog, or an entry in a damaged catalog, causes a requestor window to pop up. The requestor window gives you the opportunity to repair the damaged catalog.

The REPAIR statement in the DATASETS procedure can be used to restore a damaged catalog if you are using the SAS System in a noninteractive mode.

When specific entries within the catalog are flagged as damaged, the repair process attempts to restore only those entries. Some entries within the restored catalog may not include the last updates that occurred before a system crash or I/O error. The repair process issues warning messages for entries that could not be fully restored. When the entire catalog is flagged as potentially damaged, the repair process attempts to restore all catalog entries.

Entry Fragmentation

When a lot of update activity is done against a catalog entry, that entry eventually becomes fragmented. You can use the STAT option in the CONTENTS statement of the CATALOG procedure to determine which, if any, entries within a catalog are fragmented, as in the following

```sas
options ls=125;
proc catalog c=lilref.catalog;
contents stat;
run;
```

The following is a portion of the output that the STAT option produces for catalog entry HELP.CBT:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Page</th>
<th>Block</th>
<th>Num of Pages</th>
<th>Last Page</th>
<th>Last Block Size</th>
<th>Last Block Bytes</th>
<th>Size</th>
<th>Size Blocks</th>
<th>Block Size</th>
<th>Block Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELP</td>
<td>CBT</td>
<td>6144</td>
<td>1024</td>
<td>6</td>
<td>2</td>
<td>23040</td>
<td>23040</td>
<td>23040</td>
<td>200</td>
<td>285</td>
<td></td>
</tr>
</tbody>
</table>

To measure fragmentation within the HELP.CBT entry, let

- `page size/block size = MBP` (maximum number of blocks that can be packed on a page).
- `num blocks/MBP = minimum number of pages needed if blocks are packed optimally on pages`.

If the number of pages the entry is spread across (labeled Pages in the STAT option output) is much greater than num blocks/MBP, then the entry is fragmented.

If you apply the above formulas to the statistics shown for HELP.CBT, you get these results:

- `MBP = 6144/1024 = 6`.
- `num blocks/MBP = 6/6 = 1`.

Because the entry's data blocks are spread across five pages (even though all of the data blocks could be packed into one page), the HELP.CBT entry is considered fragmented.

To correct a fragmented catalog, use the CATALOG or COPY procedure to make a fresh copy of the catalog.

Reducing Catalog Size

Sometimes it is possible to reduce the number of data pages in a catalog by reblocking entries, eliminating wasted space in entries, or both. Reducing the number of pages in a catalog entry reduces the disk storage requirements for the file and will likely reduce the number of I/O operations required to read the data for that entry.

Normally, catalog entries created by the SAS System make efficient use of space. But the larger the catalog's page size, the greater the potential for wasted space for some entry types. You may want to use the methods described next for analyzing and reducing space requirements in heavily used catalogs that are relatively stable.

The NOWASTE Option

The following is a portion of the output that the STAT option produces for a catalog entry called SMALL.MODEL. (Earlier, you looked at a smaller subset of the STAT option output for the entry HELP.CBT).

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Page</th>
<th>Block</th>
<th>Num of Last</th>
<th>Last Pages</th>
<th>Last Block Size</th>
<th>Last Block Bytes</th>
<th>Size</th>
<th>Size Blocks</th>
<th>Block Size</th>
<th>Block Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALL</td>
<td>MODEL</td>
<td>23040</td>
<td>23040</td>
<td>1</td>
<td>2</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

In this hypothetical example, the page size of the catalog and the block size of the entry SMALL.MODEL are 23040. (23040 is the largest default block size that can be set for a 3080 disk device on the MVS host). The column headed Last Block Bytes contains the number of bytes occupied by data in the last data block for the entry. The column headed Last Block Size shows the size of the last data block for the entry. If the size of the entry's last data block is much greater than the number of bytes in the entry's last data block, then the NOWASTE option in the CATALOG procedure's COPY statement can be used to reduce the wasted space in the last data block. The NOWASTE option truncates the last data block for the entry to the smallest size possible. The unused portion of the last data block becomes a free block.

The following SAS statements copy the entry SMALL.MODEL to the catalog OUTLIB.CATALOG, turning on the NOWASTE option for the entry:

```sas
proc catalog;
copy in=inlib.model out=outlib.model nowaste;
select small model;
run;
```

The NOWASTE option truncates the last data block for the entry to the smallest size possible. The unused portion of the last data block becomes a free block.

The following SAS statements copy the entry SMALL.MODEL to the catalog OUTLIB.CATALOG, turning on the NOWASTE option for the entry:

```sas
proc catalog;
copy in=inlib.model out=outlib.model nowaste;
run;
```

The NOWASTE option truncates the last data block for the entry to the smallest size possible. The unused portion of the last data block becomes a free block.

The following SAS statements copy the entry SMALL.MODEL to the catalog OUTLIB.CATALOG, turning on the NOWASTE option for the entry:

```sas
proc catalog;
copy in=inlib.model out=outlib.model nowaste;
run;
```
The following is a portion of output that the STAT option produces:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Page</th>
<th>Block</th>
<th>Num of Blocks</th>
<th>Last Page</th>
<th>Bytes</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALL MODEL</td>
<td>23040</td>
<td>23040</td>
<td>1</td>
<td>200</td>
<td>256</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The last data block was truncated to a size of 256 bytes and the remainder of the page (22784 bytes) is now reusable space.

Many entry types are created by the SAS System with the NOWASTE option turned on. But for those entry types that are not, you should turn on the NOWASTE option only for entries that will not undergo further updating.

Reblocking Entries

When a catalog has a large page size, turning on the NOWASTE option does not always eliminate wasted space in an entry. In such a case, you may want to reblock the entry's data.

The following is a portion of the output that the STAT option produces for a catalog entry called MY.LETTER:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Page</th>
<th>Block</th>
<th>Num of Blocks</th>
<th>Last Page</th>
<th>Bytes</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MY LETTER</td>
<td>23040</td>
<td>23040</td>
<td>1</td>
<td>3000</td>
<td>2048</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Even though there are only 3000 bytes of data for MY.LETTER, the entry is using a block of size 23040.

Turning on the NOWASTE option does not reduce the wasted space in this entry. The only available entry block size greater than 2048 is equal to the catalog page size. In this case, the size of MY.LETTER can be reduced by reblocking the entry to a block size of 2048.

The following SAS statements copy the entry MY.LETTER to the catalog OUTLIB.CATALOG, changing the entry's blocksize from 23040 to 2048.

```sas
proc catalog; copy in=model out=outlib.model blksize=2048; select my.letter; run;
contents outlib.model stat; run;
```

The data for MY.LETTER now requires only 3072 bytes—one 2048-byte block and one 1024-byte block.

CONCLUSION

Version 6 SAS catalogs are used to store a variety of information essential to Version 6 of the SAS System. Part I of this paper explained the architecture of the Version 6 catalog by examining each of its components: the catalog directory, the catalog header, the subdirectory records, the catalog's data blocks, and the bitmap used for space management. Examples of catalog processing were examined to determine their memory and I/O requirements within the SAS supervisor layer. The algorithms used to manage free space within a Version 6 catalog were also discussed.

Part II of this paper discussed methods for detecting and correcting catalog fragmentation. Options for optimizing use of space within catalogs, especially catalogs that have a large page size, were presented.

REFERENCES


PRIMOS is a registered trademark of Prime Computer, Inc.

SAS, SAS/AF, SAS/FSP, SAS/GRAPH, and SAS/IML are registered trademarks of SAS Institute Inc., Cary, N.C.

VMS is a trademark of Digital Equipment Corporation, Maynard, MA.