CLASSROOM NOTES ON VSAM FOR SYSTEM PROGRAMMING - COURSE A3754

REVISED - JANUARY 1979
- MARCH 1979
- JUNE 1979

Dick Yezek
Sr. Marketing Support Representative
Central Region 10
Minneapolis, Minn.

Location:
611 E. Wisconsin Ave.,
Milwaukee, WI., 53202

JUNE, 1979

This edition contains major rewrites to the first two sections, i.e.,
Structure of a VSAM Catalog
VSAM Sharing in OS/VS.

If you have a previous edition of this paper, discard it and use only this one. Because this paper represents a set of three papers consolidated into one, the page numbering is a bit awkward. Here is a list of what you should have:

Product Notes	1 page.
Introduction	1 page.
The Structure of a Catalog	25 pages, numbered 1 to 25.
Sharing in OS/VS VSAM	40 pages, numbered 1 to 40.
VSAM Backup & Recovery	Part 1 - 17 pages numbered 1 to 17.
	Part 2 - 5 pages numbered 1 to 5.
	Part 3 - 17 pages numbered 1 to 17.
	Part 4 - 2 pages, numbered 1 and 2.
	Part 5 - 7 pages, numbered 1 to 7.

Because of this numbering scheme, problems in reproduction have been known to occur, so please check to be sure you have all the pages.

Product Notes.

This course covers the VSAM product currently available in VS1, SVS, and MVS systems. It does NOT include a discussion of the VSAM Program Product 5746-AM2 announced for the DOS/VSE System. he program product is commonly known as VSE/VSAM and was announced on January 30, 1979.

The following features of VSE/VSAM are not in the OS/VS VSAM system:

- Support of the 3310 and 3370 DASD drives, which use a new fixed block architecture.

*** - Support of new physical record sizes from .5K to 8K in multiples of .5K.

*** - Support of the NOIMBED feature for catalogs.

*** - Writing portable copies of VSAM clusters and volumes in CI mode.

- A new AMS command, CANCEL, which can cancel the job or job step.

- The ability to specify that space remaining on a volume is to owned by VSAM.

- The ability to DEFINE a cluster in a catalog without allocating any space to it.

- The ability for OPEN/CLOSE to allocate/deallocate space to a cluster defined in a catalog without space.

- Improvements in SHAREOPTIONS(4,4) that reduce path lengths and use system provided integrity for files shared across partitions, regions, or address spaces, and in loosely-coupled environments, between CPUs.

Items marked with (***) represent loss of compatibility between a DOS/VSE system and VS1, SVS, or MVS. To process any one of the files described with (***), the user will have to rebuild the item in a DOS/VSE system using options that are compatible with VS1, SVS, or MVS; IBM does not provide any conversion aids at the time of this writing.

NOTES FOR VSAM FOR SYSTEM PROGRAMMING (A3754)

This handout represents a collection of notes on VSAM I wrote over several years dealing with various aspects of VSAM, the most important of which are sharing and backup and recovery. To read this, you need to know the basic principles of VSAM, i.e., the functioning of KSDS, ESDS, and RRDS clusters and the AMS commands that work with them. These prerequisites can normally be met by attending course A3750 - VSAM Coding in OS/VS, or course A3681 - VSAM-USING ACCESS METHOD SERVICES, the VSAM Concepts and Access Method Service Usage ISP, or equivalent experience. In addition, some familiarity with Assembler Language is essential to understand some of the arguments advanced. Please note that the current (January, 1979) course does not include Global Shared Resources (GSR), Local Shared Resources (LSR), or Control Blocks in Common (CBIC). They are included in the sharing section to remind the reader that there are other ways of doing sharing if you decide that the more common options do not fit your situation.

While the notes are grouped into sections, I have deliberately omitted an index because I want you to read all of these notes, if possible while you are here in class. In my experience, the most dangerous VSAM programmer is one with a little knowledge, so I want you to read all the notes, even if you think you are 'never' going to use that; I have heard that so many times before.

I have included some 'war' stories. They are neither contrived nor meant to scare you off; they are meant to illustrate the bizarre logic that is implemented when, for one reason or another, sufficient VSAM knowledge is not at hand.

About resources. You should always have access to the latest IBM publications dealing with VSAM, including the PLMs and microfiche if necessary. I have observed that in many accounts, this need is not recognized. If there is any question about the currency of your materials, ask your IBM representative to check your SLSS - System Library Subscription Service - list. SLSS is a service available to users of IBM CPUs, IBM Program Products, and other IBM equipment. Basically, SLSS means that IBM will ship you new publications, in the quantity needed, without your ordering them. Most of the manuals are free of charge to IBM customers, but there are specific exceptions. Again, check with your IBM representative for details about your account.

Most larger accounts maintain a complete set of both PLMs and microfiche. Both are rather expensive so individuals do not normally have their own sets of these things. If you are going to be the person in charge of VSAM however, you should have your own set of VSAM PLMs.

It is best to inform your IBM System's Engineer and Program Support Representative about your plans for VSAM. They have the latest information on announcements past the publication date of these notes and information about APARs that might temporarily restrict your use of VSAM. If you are a new customer, you should remember that while IBM will assist you in determining your VSAM problems and make specific recommendations, we do not write code unless contracted to do so through custom contracts Data Service.

Finally, this is not a course in IMS or CICS. We do not cover VSAM usage in those products, or any other IBM products. While this course will help you solve VSAM problems in general, it is not designed to cope with another product's specific problems. If you assumed that, you are in the wrong course and you should call your manager and explain the situation immediately.

THE STRUCTURE OF A VSAM CATALOG

Amplification of this material can be found in the MVS Catalog Management PLM - SY26-3826. Most of the material can also be located in the VS1 PLM - VSAM Logic, SY26-3841. However, since VS1 does not support all of the VSAM catalog options, users considering migrating to the MVS system in the near future should obtain MVS manuals and use them in planning their catalog requirements.

Disclaimer.

These notes are intended for students attending the VSAM for System Programmers class, course code A3754. The purpose is to present information that is found in many current IBM publications in a single paper to facilitate reading assignments in the course. The paper is not intended to be comprehensive. Rather, it serves to allow a student to quickly assimilate the vocabulary of the catalog and some of the philosophy behind it's design. After reading this paper, serious students responsibile for VSAM catalog implimentation at their organizations will undoubtably consult relevant SRLs and PLMs in the field.

Basic Thoughts.

The catalog is really a specialized KSDS cluster. In fact, it has an index component, data component, and cluster record just like a KSDS. When I first heard that, I thought to myself: "Is a catalog cataloged in some other catalog? If so, where is the first catalog cataloged?" The answer is that no, a catalog is NOT cataloged in some other catalog; that's why I used the word 'specialized' in the first sentence in this paragraph. Now this takes some explaination, especially in MVS, because if any of you have ever done a LISTCAT ALL, you know that user catalog information prints if you direct the LISTCAT to the master catalog.
Before we get to that, there is the question of VSAM volume ownership to settle. You all know that if you DEFINE SPACE on a volume with no VSAM space on it, the space is allocated via DADSM, but the FMT1 DSCB in that volume's VTOC will contain a name of the form:

Z9999992.VSAMDSPC.Taaaaaaa.Tbbbbbbb

where the aaaaaaabbbbbbb is the timestamp value. The FMT1 DSCB also has it's password protect bit turned on (even though there is no password), but more importantly, the VSAM bit is turned on in the FMT4 DSCB on the volume and a VSAM timestamp is put in that FMT4 DSCB. Finally, a volume record is written in the catalog used to do the DEFINE. A volume record is a type 'V' record in the language of VSAM catalogs. This volume record is very important because it contains a bit map that describes the whole volume to VSAM, NOT JUST THE VSAM PORTION OF THE VOLUME. A single bit in the map represents a track on the volume in question. If the bit is a '1', the track is NOT allocated to VSAM; if it's a '0', it is allocated to VSAM. Additional fields in the volume record (called Data Space Group Set of Fields) describe whether VSAM has used any of the space assigned to it on the volume. It is unlikely that all of this will fit in just one catalog record, so volume extensions, type code W, are allocated when required and chained to the type V record.

From this point on, if you try to execute any VSAM command against our volume, it will detect the so called volume ownership flag in the FMT4 DSCB. Then it will search the catalog controlling the operation for the 'V' and 'W' records. If it finds them, the command will proceed; if not, the command will be rejected on the grounds that you have violated VSAM volume ownership! Simply stated, VSAM volume ownership means:

- The FMT4 DSCB on the volume has the VSAM ownership flag turned on.
- The catalog in question has a type 'V' record in it with the volume's serial number.

If any other command using a different catalog is executed against our volume, it will be rejected because the catalog will not have a matching type 'V' record.

Now, back to the question of who owns a catalog. If you try to first DEFINE SPACE on a volume and then put a catalog in it, you can't because a catalog already owns the space by the rules described above. You can't just DEFINE a KSDS cluster either because that requires VSAM space (except for UNIQUE clusters), and I just covered that. That leaves two possibilities; either the catalog owns itself, or it is owned by the master catalog.

Now if the catalog is owned by the master catalog, we should expect to find type 'V' and 'W' records in MCAT. You will not. Therefore, the catalog must own itself! This makes sense because it solves the problem of what to do with the ownership of the master catalog - MCAT owns itself also!

How a Catalog Is Defined.

To solve the problem of how you place a specialized KSDS cluster (a catalog) on a volume that is not now owned by VSAM (no space defined on it), the AMS commands DEFINE MCAT and DEFINE UCAT allocate both VSAM space and a catalog in that space at the same time. There are two basic ways of doing it:

1. DEFINE UCAT(.....CYL(P,Q).....)

2. DEFINE UCAT(.....CYL(P,Q).....) -
   DATA(.....REC(A,B).....)

In method one, the amount of space allocate to VSAM is 'P' cylinders and all of it is used for the catalog.
In method two, the amount of space allocated to VSAM is still 'P' cylinders, but only 'A' records of it form the catalog. The reason record allocation is chosen is that a table appears in the AMS manual which defines the size of a catalog in records; if you use the DEFINE shown, you put the output of the table calculation directly in the DATA component.
You can also DEFINE the index component separately if you wish, but you will see later that it doesn't have much effect on the size of the index; if you code the index component, it's size is simply added to the amount you specified in the data component and VSAM goes right ahead and DEFINEs the catalog according to it's internal specs.

The AMS manual also contains some formulae for estimating the size of the secondary allocation. More about that later.

When the DEFINE is finished, you will find the following has been done:

- Space will have been allocated on the volume. It will have the usual FMT1 identifier in the VTOC, but the first qualifier will be Z9999994 instead of Z9999992. The '4' indicates a catalog is present in this space.
  THE CATALOG ALWAYS OCCUPIES THE FIRST TRACKS IN THE SPACE.

- The first three records in the catalog will be a type D for data component, type I for index component, and type C for cluster component. These three form the definition of a KSDS cluster, in this case the catalog itself. Among other things, these records contain the name of the catalog and it's passwords if any are used.

- The fourth record in the catalog (numbered 3, VSAM starts counting catalog records at zero) will be a special type L record - the catalog control record or CCR for short. It is very important because VSAM commands that manipulate catalogs (like REPRO) check the format of this record to determine if the object being processed is indeed a catalog.

- The next 5 records in a catalog are type E - extensions of the cluster definition. More about these later.

- Starting at the tenth record, you will find a type V record and the necessary type W extensions. These describe the volume upon which the catalog resides. They also prove that the catalog owns itself!

The sum of all these records is termed the catalog's Self Describing Records - SDR. Just to make sure you got it, let's assume you are searching for a catalog and have located the volume where you think it should be. You search the VTOC for a FMT1 DSCB that has a first qualifier of Z9999994. If you find it, it gives the extent of the space. You then read the fourth record in the extent and it should be a type L catalog record. If it is, you use all the other SDR to establish the proper catalog control blocks and OPEN the catalog.

One last point. Why does the LISTCAT to the MCAT list user catalog information? Because one of the final things done in a DEFINE UCAT is to put a single entry in MCAT called a type U - user catalog record. A type U record NEVER implies VSAM volume ownership - it just informs the user that a user catalog is expected to be on volume such and such. By the way, you can get a type U record placed in MCAT by doing an IMPORT CONNECT. This is one of the few VSAM commands that is not checked; VSAM simply takes you word that the designated volume contains a catalog.

Internal Structure.

Now that the ownership and MCAT type U record have been explained, I can begin to tell you about those 5 special records in the SDR I skipped over previously. Now the catalog is a KSDS cluster, but it is physically laid out defferently. It has three parts, allocated in this order:

- The low key range - LKR - which occupies about 90% of the space allocated to the catalog.
- The index set. This is always one, and only one, control area.
- The high key range -HKR - which occupies about 10% of the space allocated to the catalog.

Every record in a catalog is a keyed record, and the key always is in the first 44 bytes of the record. The catalog then is really a keyrange KSDS cluster where the two ranges are defined as follows:

LKR	x'00000000.....00000000'	total of 44 bytes
	x'3FFFFFFF.....FFFFFFFF'	
HKR	x'40000000.....00000000'	
	x'FFFFFFFF.....FFFFFFFF'	

The idea here is simple. A data set name, VSAM or nonVSAM, must have at least one character to be coded on a JCL statement. That character collates higher than x'40' which is a blank. Thus a data set name will always be placed in the HKR. The LKR contains all the information in the catalog about the data set. When a name is cataloged, a LKR record is selected from a free chain maintained by catalog management and the information is formatted and written. Now a LKR record is ALWAYS a full CI in size, so to find it later, all you need to know is the CI number. The CI number is tacked on to the HKR name entry as a 3 byte field and a total of 47 bytes is inserted into the HKR. We now have a HKR record that has a 44 byte name and a 3 byte CI number. The catalog information about that entry is in the LKR at the CI number specified in the HKR entry!

All CIs in a catalog are 512 bytes, regardless of circumstances. Therefore, a single entry in a catalog requires a 47 byte HKR entry and a 512 byte LKR CI for a total of 559 bytes. The size of common catalog entries is:

KSDS	Three entries, cluster, data, and index. (1679 bytes) Names omitted are supplied by VSAM.
ESDS	Two entries, cluster and data. (1118 bytes) Names omitted are supplied by VSAM.
RRDS	- same as ESDS. -
nonVSAM	One entry - 559 bytes.

This neatly explains the 90 - 10 ratio of LKR to HKR; exactly 10 records can fit in the CI of the HKR so for every 10 LKR records you need exactly 1 HKR CI. (10% of 10 is 1.) The index needs more work.

For a catalog, the IMBED option is forced. Now IMBED imbeds and replicates the sequence set of the index in the first track of every data component control area. This leaves only the index set of the index component to be explained. Since a catalog cannot span volumes, the index set is expected to be very small; in many cases it is only a single CI. Since one track of all current supported devices holds many 512 byte blocks, one control area for the index set seems ample; after all, the index set is only higher level index CIs. For a catalog, THE INDEX SET IS ALWAYS A SINGLE CONTROL AREA.

This brings use to the size of the control area. It is forced, just like the CISIZE, but is device dependent:

CA Size In Tracks	Device
3	3330
3	3350
3	2305 Model 2
4	2305 Model 1
5	3340/3344
5	2314

Just for fun, let's allocate a one cylinder catalog on a 3350. Now a control area is also known as an Allocation Unit - AU in VSAM lingo because a control area is the smallest quantity that can be allocated during a CA split. Our 3350 holds a total of 10 AUs in one cylinder.
One AU is always set aside by VSAM for the index set, leaving 9 for the data component.
The HKR is 10% of the data component. 10% of 9 is .9, rounded high yields 1.
The remaining 8 AUs form the LKR.

Looking at it another way, there are a total of 9 AUs for the data. But the sequence set is always imbedded in a catalog, so one track of every data AU is used for the sequence set. In the case of the 3350, that leaves two tracks per AU or 18 out of the original 30 tracks for actual data. Extending this to a larger catalog, you see that about 1/3 of the catalog space is used to house the index and 2/3 is used for data. This means that VSAM catalogs are considerably larger than the older SYSCTLG structure, now simply termed CVOLs - control volumes. Recalling that a nonVSAM entry in a catalog takes 559 bytes, and that a simple entry in a CVOL might take only 60 bytes, a VSAM catalog used only for nonVSAM entries can take 10 times the space required for CVOLs plus another 1/3 for the VSAM index!
This fact alone MUST NEVER be the sole determinator of the type of catalog you use - backup and recovery becomes quite important as well as utility support for the catalog you eventually choose. Most customers wind up with some of both kinds, but you should 'prove' to yourself that the VSAM catalog will not do. More about this later.

Review.

A catalog is a specialized key range KSDS cluster with the CISIZE forced to 512, the key always 44 bytes at offset zero, and the CA size forced to 3 tracks for the common 3330/3350 devices.

The catalog is always composed of a LKR (90%), index set (one CA), and a HKR (10%) allocated in that order.

All LKR records occupy a full CI (sometimes more than one); a HKR record is 47 bytes and the the data set name (44 bytes) and the CI number of the associated LKR record (3 bytes).

Unexplained so far are the five special records in the catalog's SDR. Remember, the first three were the type D, I, and C records that defined the catalog as a cluster and the fourth was the type L, the catalog control record. Remembering that the CI numbers in the LKR start with zero, we have:

CI Number	Description.
0	Type D, describes the data component.
1	Type I, describes the index component.
2	Type C, cluster definition of the catalog. The catalog's passwords are here.
3	Type L, the catalog control record (CCR).
4	Type E, an extension of CI #1, this record describes the location of the index set.
5	Type E, an extension of CI #0, this record describes the LKR.
6	Type E, an extension of CI #1, this record describes the sequence set imbedded in the LKR
7	Type E, an extension of CI #0, this record describes the HKR.
8	Type E, an extension of CI #1, this record describes the sequence set imbedded in the HKR
9	Type V, the volume record for the volume upon which this catalog resides.
10 - 13	Type W, these are extensions to CI #9 as required by the device type.

The reader should understand that the three parts of the catalog can be printed quite easily using the PRINT command of AMS. Since the FROMKEY parameter of that command supports generic keys, you can use it to print a specific HKR record by coding:

```
PRINT   IDS(PAY.VSAMCAT.X/PASSWORDX)FKEY(MY.DATA.SET)COUNT(1)
```

The IDS above is an MVS exclusive and would be coded as IFILE in VS1 and SVS. Once you read the result of this printout, you can print the associated LKR record by coding:

```
PRINT   IDS(PAY.VSAMCAT.X/PASSWORDX)FKEY(X'00xxxxxx')COUNT(1)
```

The 'xxxxxx' in the previous coding is the CI number obtained from the HKR. The reason for the leading byte of zeros is that all LKR records are formatted that way to force them in the LKR even if some error in formatting should occur. You can print the SDR by omitting the FKEY and specifying a count of 14 - the COUNT parameter starts at one, not zero. If you wanted to print just the CCR, you could specify:

```
PRINT   IDS(PAY.VSAMCAT.X/PASSWORD)SKIP(3)COUNT(1)
```

It should be obvious to you that in all the above examples, PAY.VSAMCAT.X is the name of your catalog and PASSWORD in the catalog is the master level password. If you think you have had just about enough of the SDR for the time being, consider this actual problem.

A customer fired a system programmer who knew the master password to MCAT and all UCATs. This programmer then used the ALTER command of AMS to change all the passwords to eight hexadecimal characters of the form x'0001020304050607'. These are unprintable on most print train arrangements. Of course no AMS command would print or list those passwords in hex because to print or to list the passwords of a catalog, you must know the master password. The account spent three whole days trying to find out what went wrong and how to fix it. How would you fix it?

Answer.
First, do a LISTVTOC of the volumes in question to locate the FMT1 that has Z9999994 as the first qualifier. In VS1 and SVS, the master catalog will have Z9999996 as the first qualifier. You now know the location of the catalogs on their volumes. The master password of each catalog is in CI #2. You can use an old listing of the catalogs to determine the exact location of the master password in the CI if you have such a listing; if not, you will have to use the PLMs (or the information you learn in this class) to figure out the layout of a type C record. (It's not too bad, but you should try to figure one out BEFORE you have to do it.) Then, you will have to SUPERZAP the master passwords

in each of the catalogs to a known combination. The ALTER command of AMS can be used to change the rest of the passwords.

Note: You can't use the master password of MCAT to ALTER passwords in a UCAT; you must know the UCAT's passwords to ALTER them. This is also true of LISTCAT and PRINT.

Note2: See page 18 in this section for some other ways to 'fix' a catalog in today's environment.

How the Catalog Uses Its Space.

You already know that the SDR always occur first in a catalog and that the index set is physically between the LKR and HKR. You may think that catalog management simply continues to assign LKR CI numbers in ascending sequence as needed. Such is not the case. In a new catalog, the LKR CI's that remain after the SDR are generated are all chained together by a preformat routine and are called 'free' records. The chain is high to low, and the chain is anchored in the CCR at offset x'36'. The number of free CI's in the LKR is in the CCR at offset x'33'. Note that the exact usage of the CCR has changed slightly from the early release of VSAM, i.e., at one time, the chaining described above was not done unless the CI was available due to deletion. The Catalog Management PLM still describes the old method of using the CCR. In the present system, LKR records will be assigned from the high CI's down to the low ones because of the way the free records are chained together. The reason for the chain being in the order described is to minimize seek time between the index set, HKR, and LKR. The HKR must be used in ascending sequence, but because the index set is physically between the HKR and LKR, the free record chaining technique will tend to group the records used daily physically close together.

If you analyze the CCR at offset x'2D' and x'30', you will find that it contains two fields: the number of the highest CI number in the first extent of the LKR and the number of the 'next' free CI that was never assigned (preformatted). What these two fields actually do is to reserve a few CI's for use by the EOV routines in case of errors encountered in extending the catalog. The reserved CI's are not preformatted nor are chained together themselves; they can be located by using the CCR without a chain because they are always physically adjacent. The number reserved is currently either 2 or 3 depending on the PTF level of VSAM.

Master Catalog vs. User Catalog.

The internal structure of MCATs and UCATs is identical. A minor difference in MVS systems is that all catalogs will have the prefix Z9999994 in their FMT1 DSCB because all catalogs in MVS are generated on a system that already has VSAM active (i.e., MVS), and since there cannot be two MCATs in a single system, a DEF MCAT in MVS will be changed to a DEF UCAT automatically. In VS1 and SVS, VSAM is optional, so at the time VSAM is installed, the driving system theoretically does not have VSAM active, so the DEF MCAT generates a Z9999996 prefix in those systems. (It really doesn't matter if VSAM is active in VS1 or SVS; a DEF MCAT in those systems generates the Z9999996 prefix.)
THIS HAS NO BEARING ON CATALOG INTERCHANGABILITY IN OS/VS SYSTEMS!
In MVS, any catalog can become the MCAT, regardless of the prefix, because the MCAT is designated via a member of the SYS1.NUCLEUS data set (SYSCATLG) rather than by a prefix.

Far more important than the Z999999x qualifier is the content of the proposed MCAT. VS1 and SVS have no MCAT requirements other than the fact that one must exist if you want to use VSAM, but MVS does have special requirements for its MCAT. This affects the backup and recovery of the MCAT.
In a VS1 or SVS system, a small MCAT can be used which contains only the UCAT connectors and the SDR; if this catalog is lost, it is a relatively simple matter to construct another MCAT on some other volume and connect the UCATs to it via IMPORT CONNECT. If the old MCAT is partially usable, it can be connected to the new MCAT in a similiar manner and data sets controlled by it can be exported. All of this is controlled using appropiate JOBCAT/STEPCAT statements.

The MVS MCAT is quite another situation. It is the system's catalog - the one that will be used by the schedular to handle the DISP=(,CATLG) and DISP=(,UNCATLG) parameters if no other catalog is specified. As such, it must be online at all times, so it is allocated at IPL. IF IT CANNOT BE ALLOCATED AT IPL, THE IPL FAILS!

Earlier, I stated that the MVS MCAT was located via the member SYSCATLG of the SYS1.NUCLEUS data set. That member contains the name, volume serial number, and device type of the MCAT. This means you cannot have a small 'duplicate' MCAT to 'get you going' in the event of a disaster unless you are willing to have an EXACT duplicate satisfying the requirements of SYSCATLG. While utilities can certainly create a backup copy of your MCAT, if you plan on having this EXACT duplicate, what will be your frequency of running your backup job in the light of the fact that the MVS schedular can update your MCAT in MVS? In addition to this, the following are also requirements for the MVS master catalog:

- All SYS1.xxxxxxxx data sets must be cataloged in it.***
- All data sets named in LINKLIST (the concatenations to SYS1.LINKLIB) must be cataloged in it, regardless of the first level qualifier of the concatenations.
- All data sets used in the procedure to start JES must be cataloged in it.

Up to this point, you might have decided that the only inconvenience might be buying another 3350 unit if you decide on those EXACT duplicate catalogs, or changing the names of the items in the linklists, but let us look further into these requirements.

The MVS Page, Swap, and SMF Data Set Problem.

MVS commonly uses at least 5 page data sets and 3 swap data sets. (These are averages - the combined maximum is 64.) Most users like to spread these out across channels and control units for performance reasons. If a user had a total of 8 of these data sets, and put each one on a separate volume, then because of the fact that these data sets are named SYS1.xxxxxxxx, and because of the fact that they must be available at IPL, and because only the MCAT is available at IPL, they all MUST be cataloged in MCAT! But they are all VSAM clusters. Therefor, the MCAT in the example now has volume ownership to nine volumes - itself and the eight page and swap data sets.

What this means is that if you want to place any other VSAM clusters on the nine volumes in question, you MUST catalog the entry in the MCAT, regardless of the fact that your naming convention dictated a UCAT, and regardless of the fact that having such entries in MCAT may complicate your backup and recovery procedures (remember that EXACT copy you were going to keep), and regardless of the fact that you may be forced to divulge the master password of MCAT to those doing maintenance on the 'other' VSAM clusters on these nine volumes.

There is one more data set involved with MVS paging - SYS1.STGINDEX - which is a VSAM cluster. It is quite small and can be placed on one of the volumes already owned by MCAT, usually on the MCAT volume itself. (It is used to support journaling of VIO data sets.)

In MVS/SE Release 2, (hereafter called SE2), all SMF data sets will be specialized VSAM clusters and will react exactly like the page and swap data sets described above. They are named SYS1.MANxx where xx is a max of 36; it is likely that no more than 4 will ever be defined by the average user. Unless you put these clusters on volumes already owned by MCAT, your MCAT volume ownership problems will be aggravated by this change to SMF.

The Space Reclaimation Problem.

Since the catalog is really a KSDS cluster, the keys in the index are compressed just like any other KSDS cluster. The keys in any index are therefore generic keys, which has great significance with respect to catalogs. In the following example, I'm going to show you logically what the sequence set looks like in a catalog, and I'm going to use decimal rather than hex to simplify things.

*** Some customers have attempted to modify VSAM so as to remove volume ownership from the MCAT. As far as I know, it works, but with limitations. Do not attemt any mods except as a last resort.

First, examine these CIs where I show only the first and last logical records.

CI #0 49.....1784.....RDF-CIDF
CI #1 2106.....3824.....RDF-CIDF
CI #2 4001.....4898.....RDF-CIDF
CI #3 5005.....5678.....RDF-CIDF
CI #4 6342 etc.

We could build a simple index as follows: (CI #4 is shown only for future development.)

CI Number	High Key in Index
0	1784
1	3824
2	4898
3	5678

This index will work. It's the type of index used by ISAM and it will insert all records between 1784 and 3824 into CI #1. Suppose we change the index in the following manner:

CI Number	High Key in Index
0	1999
1	3999
2	4999
3	5999

The keys shown are a type of generic key. They change the range of the numbers that 'belong to the various CIs - all records between and including 2000 and 3999 will go in CI #1. Why do this? Well, in ISAM, no record deletion was permitted, so the possibility of deleting the record that represented the high key in an index record didn't exist. But in VSAM, you are allowed to delete record 1784. If we left the index like the ISAM system, we would have to modify the index if a user happened to delete 1784 in my example, which implies we would ALWAYS have to check the index on any deletion because the user might have deleted a high key.
Now if we use the generic key, and ALWAYS LEAVE IT IN THE INDEX, we will never have to check the key on a delete. VSAM uses this technique. VSAM goes one step further - it compresses the key in the index. Key compression is too lengthy to describe here, so just take it on faith for the time being that the index in VSAM would appear LOGICALLY as:

CI Number	High Key in Index
0	1
1	3
2	4
3	5

This is an example of what is known as rear compression ONLY. If all the keys are generic, then you need record only the nongeneric portion; the algorithms can fill in the proper number of 9s based on how much of the key is missing. VSAM also employs front compression, for for studying the catalog, you need not understand that. For a catalog, the most important idea to grasp here is that if you delete every record between 2000 and 3999, VSAM WILL NOT CHANGE THE INDEX ENTRY!
THE ONLY WAY TO REUSE THE SPACE ALLOCATED TO THE RANGE 2000 TO 3999 IS TO INSERT RECORDS WITH KEYS BETWEEN 2000 AND 3999 INCLUSIVE.
VSAM does all of this using hexadecimal keys and the generic 'fill' character is x'FF', but the idea is the same.

Now, imagine a VSAM catalog. It has many compressed generic keys in it's sequence sets. As you use your catalog you catalog new data sets daily and delete others from time to time. Long ago you established a data set naming convention of A.B.C.date for all your data sets. This means that every new data set cataloged is ALWAYS in higher collating sequence than all previous data sets. This means that VSAM is forced to place an entry in the LAST control interval in the HKR; if it's full, VSAM is forced to do a CI split because all other CIs have compressed generic keys in the index that collate lower than your new data set.
Further, because VSAM does the insert into the HKR using direct insert strategy (split 50% of the records), the HKR control areas will have only 25% utilization presuming you delete nothing. The net result of all of this is that:

- If you get an error message that indicates your catalog is full, you can delete every last data set, but if the new data set names collate higher, the catalog will still appear full due to the compressed generic keys.

- If your data set naming convention causes new data sets to always collate higher than old ones, your HKR can fill up 4 times faster than normal. Since there is no way to control the size of the HKR (it's always 10% of the data component), you might be forced to allocate a catalog 4 times larger than you really need! (The latter assumes you don't care to reorganize from time to time.

- The is no way to reorganize a catalog 'in place', i.e., there is no way to compress a catalog like you compress a PDS.
  The only way to reorganize a catalog is to DELETE the catalog and start over!

At this point, I'm sure you are convinced that you will NOT choose a data set naming convention that includes something like the date as the only distinguishing factor.
NOTE: CNVTCAT converts a CVOL by reading it in strict ascending sequence. If you use it, you will need a HKR of 4 times the size actually required, which may mean a VSAM catalog 4 times the design size. You might be able to use the CNVTCAT command for a partial conversion however.

If you are a VS1 or SVS user, you are finished with this section, but you might as well read on to see what you are going to have to consider when you move to MVS. Now you can modify your naming conventions to fit the VSAM catalog in all respects except one - GDGs. In MVS, you don't have to use any CVOLs at all, so VSAM must allow GDGs to be cataloged in a VSAM catalog - and GDGs always have .GxxxxVyy as the last 9 characters of the name. Since the generation number increases by one for most cases, GDGs cataloged in a VSAM catalog produce the problem described above.

The GDG case is a bit more problematical because you cannot afford to reorganize the catalog every time a GDG starts using up too much catalog space, yet you have no alternative to the GDG. Of course you can specify that a GDG is to reset to G0001 after so many iterations, but what if you have already designed the GDG to increase 'forever'? You may not be able to change your design.

This represents one of the toughest questions you will have to answer - a question that arises from the internal structure of the catalog. It's tough because by having one type of catalog in your system, you can streamline recovery procedures - a very worthwile end if you have a requirement to 'get back up in so many (fill in blank _____)' after a disaster. Paradoxically, because you do that, you may have to execute those procedures more frequently because of the catalog structure.

The Catalog Search Algorithm Problem.

In VS1 and SVS, the the only way any entry is placed in the SYSCTLG (now termed CVOL) is via the JCL DISP parameter, the IEHPROGM utility, or the CAMLIST and related macros. Similarly, the only way entries are placed in the VSAM catalog is via the AMS commands. Thus there are really two separate catalog systems.
Further, if a UCAT is to be utilized, JOBCAT/STEPCAT statements must appear in order to direct VSAM to search those catalogs. Even if you use the CATALOG parameter in AMS commands, you must use JOBCAT, STEPCAT, or a separate DD statement to identify the catalog you wish used.

Let's look at the following JCL, which represents a step in some job.

```
.........
//STEPX     EXEC      PGM=IDCAMS
//STEPCAT   DD        DSN=VSAM.USERCAT,DISP=SHR
//DA        DD        DSN=VSAM.CLUSTERA,DISP=SHR
//DB        DD        DSN=NONVSAM.FILEX,DISP=SHR
//DC        DD        DSN=NONVSAM.FILEY,DISP=(,CATLG),UNIT=TAPE
//SYSPRINT  DD        SYSOUT=A
//SYSIN     DD        *
     REPRO  INFILE(DB)OUTFILE(DC)
     REPRO  INFILE(DC)OUTFILE(DA)
.........
```

All VS1 and SVS users would quickly agree that VSAM.CLUSTERA is in VSAM.USERCAT because of the presence of the STEPCAT and absence of any CATALOG directive; NONVSAM.FILEX may be in the UCAT, MCAT, or a CVOL (they will be searched in that order); and NONVSAM.FILEY will be cataloged in the CVOL because there is no facility in VSAM in these systems to handle the DISP parameter of JCL.

In MVS, you may have only VSAM catalogs if you wish, so MVS MUST be able to handle JCL dispositions. The method used for this service is to add the capability of a UCAT alias to the catalog structure. This alias entry is a single level of qualification. It is used by VSAM to 'direct' the search of catalogs based on THE FIRST LEVEL QUALIFIER OF THE DATA SET NAME , assuming you have not specified otherwise via JOBCAT/STEPCAT or the CATALOG parameter of AMS.

Suppose you place an alias of XXX in an MVS master catalog for the UCAT named USERX.VSAM.CAT. Suppose further that you execute a simple step with no JOBCAT, STEPCAT, or AMS command that uses the CATALOG parameter. If the following JCL statement appears:

```
//DX      DD        DSN=XXX.A.B,DISP=(,CATLG),UNIT=TAPE
```

the data set will be cataloged in USERX.VSAM.CAT because the first qualifier of the data set name matches the alias XXX in MCAT. Let's examine the effects of that one characteristic.

- No JOBCAT/STEPCAT is needed. At the very least, a user who formerly ALWAYS used JOBCAT/STEPCAT now needs them 'sometimes'. Just when, exactly when, are those times?

- As you might suspect, a single alias cannot point to different catalogs - only one. Suppose your present naming convention uses a unique first level qualifier for some application and you MUST split the entries among at least two catalogs to solve GDG problems? You will probably have to change your naming convention in this situation. Do you have a way of locating all occurances of that old name?

To understand the implications of this alias further, it is necessary to understand that the alias can be used for CVOLs as well as UCATs. Thus the general rule for using all catalogs, not just VSAM catalogs, in the MVS system is:

The first level qualifier of a data set name controls the catalog to be used to hold the entry.

Again, let me point out that the general rule assumes there is no JOBCAT/STEPCAT, the exact opposite of VS1 and SVS. Now if an MVS user codes the CATALOG parameter in an AMS command, it will override all other catalog specifications; that much is the same as VS1 and SVS. If an MVS user codes no CATALOG parameter and does not use JOBCAT/STEPCAT, MVS will use the alias structure in MCAT to select the catalog. IF NO ALIAS MATCHES THE FIRST LEVEL QUALIFIER, THE MCAT IS ALWAYS CHOSEN. While not the same as VS1 or SVS, this last mechandism is at least controllable via data set names. Are there more differences? Yes, but to understand them, you need to understand the MVS catalog search strategy.

MVS Catalog Search Strategy.

Searching for an entry already cataloged.

1. If the CAMLIST macro is executed directing the search to a CVOL, the CVOL is searched as long as the CVOL still appears in the MCAT. (CVOL entries in an MVS master catalog must be named SYSCTLG.Vxxxxxx. 'xxxxxx' is usually the volume serial number of the CVOL.)

2. If the CATALOG parameter is coded in an AMS command, the specified catalog is searched.

3. IF JOBCAT/STEPCAT is present, the specified catalogs are searched in the order of their concatenations. If the entry is found, the search is terminated and the MCAT is not searched; if the entry is not found, control passes to the next step.

   STEPCAT overrides JOBCAT completely for a given step. If both JOBCAT and STEPCAT are used in a step, only the catalogs named in the STEPCAT and it's concatenations will be used.

4. If the search argument is not a qualified name, go to the next step.

   If the search argument is qualified, the first qualifier is extracted and used to scan a chain of control blocks in the user's address space called the PCCB chain - Private Catalog Control Block. If a user has accessed a catalog other than MCAT already, a PCCB will exist which has a total of two names in it: the name of the catalog and the name of the alias used to locate the catalog, if an alias was used at all. (JOBCAT/STEPCAT doesn't use aliases.)

   If the argument matches the alias name or the first qualifier of the catalog name, the search stops and the catalog named in the PCCB is utilized.

   If there is no match, control goes to the next step. I might point out here that there may be multiple alias entries in the MCAT for a single catalog - CVOL or UCAT - but only one of those aliases appears in the PCCB. THIS HAS PERFORMANCE IMPLICATIONS, ESPECIALLY IN TSO.

5. MCAT is now searched.

   If the data set named is not qualified, MCAT is searched for only data set entries and the search terminates here with either a 'found' condition or an error.

If the data set name is qualified, the first level qualifier is extracted and the system tests to see if it matches either:

- a user catalog with that single level name, or
- an alias to a user catalog, or
- an alias to a CVOL.

If any of these are true, the true name of the catalog is extracted and matched against the existing PCCB chain. If no match is found, VSAM goes to the MVS scheduler with a request to dynamically allocate the catalog. This means that a new PCCB will be added to your chain of existing PCCBs and it will contain the alias name as well as the true name of the catalog if an alias was used to locate it; if an alias was not used to locate the catalog, the PCCB will contain just one name, the true name of the catalog.

If a match is found on an existing PCCB, the existing PCCB indicates the required catalog is already allocated to the job step, so it is simply searched.
ADDITIONAL NAMES ARE NOT PLACED IN THE PCCB; THE PCCB HAS ROOM FOR TWO NAMES ONLY. ALL OTHER ALIAS SEARCHES MUST GO THROUGH THE MCAT!

Exceptions:

1. As stated above, if a UCAT has a name AAA (single level), and you are searching for data set AAA.QQQ, the proper catalog will be found.
   If the name of the catalog is P.D.Q with an alias of AAA, the catalog will also be found.

   However, if the name of the catalog is AAA.ZZZ(qualified name whose first qualifier matches the argument of the search), the CATALOG WILL NOT BE FOUND. You must supply JOBCAT, STEPCAT, or the CATALOG parameter in this case.
   This is a performance consideration because if it was allowed, there could be hundreds or even thousands of catalogs with the first qualifier of AAA, and all would then have to be searched before an error was declared.

2. VSAM will permit you to catalog a GDG named A in the UCAT named A. In this situation, it is obvious that the SDR of the UCAT contain a cluster record with the name A. This seems like allowing duplicate names in the same catalog. Just remember that the true names of GDG entries will be A.GxxxxVyy however which does not conflict with the name A. (Internally, there will be a GDG base record with the name A which will have associations with nonVSAM entries named A.GxxxxVyy. The exception is allowing the GDG base in a catalog with the same name.)

Exceptions that affect both algorithms.

1. You cannot put the MCAT in a list of concatenated DD statements to JOBCAT/STEPCAT, nor can you name MCAT on the JOBCAT/STEPCAT itself.
   While the DD statement containing the name of MCAT will be allocated, VSAM makes internal checks on the name of the catalog. If it is the name of the master catalog, the position of the DD statement in the user's jobstream is ignored and MCAT is searched according to the rules stated above.

2. For an open catalog, the latest statistics are kept in a block allocated in common storage called the CAXWA. It's contents are not written back to the catalog until VSAM is 'finished' with the catalog, or until the catalog must be extended. For a simple UCAT, VSAM finishes with it at end of step.
   This implies a TSO user will have to log off and on again before certain catalog operations like changing alias entries are known to other users.

Searching for a catalog in which to place an entry.

1. If the CAMLIST macro is executed directed to a CVOL, the CVOL is used.

2. If the CATALOG parameter is used in an AMS command, the named catalog is used.

3. If a JOBCAT or STEPCAT is present, the named catalog is used.

   Note that the JOBCAT/STEPCAT catalog is used EVEN IF THERE IS AN ALIAS IN MCAT POINTING TO A DIFFERENT CATALOG!

   This is quite different from the search used in locating an entry. If a duplicate name is found in the JOB/STEPCAT, the catalog operation will fail, even though there is an alias in MCAT that could have been used to find some other catalog.

4. If none of the above are true, then MCAT is searched as follows:
   If the name being inserted is NOT qualified, it is cataloged in MCAT, barring some other error such as duplicate name or lack of proper password.

   If the name is qualified, extract the high level qualifier and proceed to the next step.

5. The qualifier is checked to see if it matches:
   - a user catalog with a single level name, or
   - an alias to a user catalog, or
   - an alias to a CVOL.

   If any of these situations are true, a check is made on the PCCB chain in your address space to determine if you have already allocated the required catalog. If you have, it is used. If not, the scheduler is used to dynamically allocate the catalog as described previously.

Exception.

If you are cataloging a GDG, VSAM realizes it must locate the correct GDG base record. Even though you specified a JOBCAT or STEPCAT, VSAM will still perform the 'extended' search described on pages 11 and 12 in an effort to locate the correct GDG base.

Where Are We Now?

Recall that I was discussing the MVS catalog search algorithm problem, in the middle of which I digressed somewhat to explain the MVS search algorithm. Turn back to page 10 and reexamine the JCL shown there. Look carefully at statement DC. In VS1 and SVS, the data set NONVSAM.FILEY was cataloged in a CVOL because those systems have two separate catalog structures; in MVS, by item #3 above, that same data set will be cataloged in VSAM.USERCAT. Even if there is a duplicate entry in VSAM.USERCAT, MVS will not go to the MCAT and search for an alias or anything else; in that case, MVS will return an error.
Notice that the JCL does not work the same way so you are left with either changing the JCL or changing the naming conventions. This was brought about by the fact that:

IN MVS, THE DISP=(,CATLG) PARAMETER OF JCL WILL BE AFFECTED BY THE PRESENCE OR ABSENCE OF A JOBCAT OR STEPCAT STATEMENT.

IN MVS, THERE IS NO FUNCTION EQUIVALENT TO JOBCAT/STEPCAT FOR CVOLS.

The worst possible situation is a customer that must maintain two jobstreams, one for an MVS machine and the other for a nonMVS machine. There are several ways of dealing with the problem, and I am going to show you some of them, but the point is, you must deal with it - it will not just go away.

Perhaps the best way is to disallow the use of the DISP=(,CATLG) parameter of JCL except for GDGs (where MVS makes an exception anyway). This implies that you have some 'group', probably your 'data base' group, that is responsible for all allocations that require cataloging. The group does all the allocations and so there is no need for the programmer to code DISP=(,CATLG). This also makes control of data set naming conventions better.
One problem that you must address with this method is how are you going to test new JCL? Application programmers 'integrating' components for the first time have a legitimate claim to putting new entries in a catalog, yet the entries may be very short lived. This situation may be unwieldly unless the application programmers have instant access to the catalog's control group. If you are forced to do testing at odd hours, this idea may be unworkable.

Another approach can be to code DISP=(,PASS) and do the actual cataloging in a future step or you can elect to set up previous steps to do the cataloging. Thus, you could recode the example of page 10 as:

```
.........
//STEPXP     EXEC       PGM=IEFBR14
//DC         DD         DSN=NONVSAM.FILEY,DISP=(,CATLG),UNIT=TAPE
//STEPX      EXEC       PGM=IDCAMS
//STEPCAT    DD         DSN=VSAM.USERCAT,DISP=SHR
//DA         DD         DSN=VSAM.CLUSTERA,DISP=SHR
//DB         DD         DSN=NONVSAM.FILEX,DISP=SHR
//DC         DD         DSN=NONVSAM.FILEY,DISP=SHR
//SYSPRINT   DD         SYSOUT=A
//SYSIN      DD         *
     REPRO   INFILE(DB)OUTFILE(DC)
     REPRO   INFILE(DC)OUTFILE(DA)
.........
```

This approach adds more overhead in that you have extra steps. Also, if the cataloging is done after the data set is passed, you can complicate the recovery considerations for the job because if the job fails before the cataloging takes place, you can have difficulty finding out where the allocations are at the time of the failure. Furthermore, you will either have to provide for 'customizing' the recovery JCL 'on the spot' because volume serial numbers will be needed in the JCL, or you will have to execute IEHPROGM and/or AMS to force the required entries in the catalogs before the recovery JCL is executed.
Another consideration is that there are certain situations (rare) in MVS where cataloging will not occur unless you OPEN the data set. These situations involve tape data sets on dual density tape drives where the density is not specified in the JCL. (The cases are fully described in the MVS Conversion Notebook.) In those instances, IEFBR14 would not work as a method of precataloging, but you could still use IEFBR14 as a 'post processor' as you as you opened the data set in some previous step.

Once the JCL is set up in this fashion however, it will function correctly in any system, without the need for special accounting exits. This key advantage may far outweigh any of the disadvantages mentioned above.

Still a third idea involves writing a routine that scans the JCL as it comes in and disallows any JOBCAT or STEPCAT statements. (The exit is described in the SMF manual and is also referenced in the JES2 or JES3 System Programmer's Guide.) The normal technique used in the exit is to set the DSN on JOBCATs and STEPCATs to DSN=NULLFILE which has the same effect as DUMMY.

This technique can also be debated. On the 'pro' side, it is certainly automatic, requires no extra steps, enforces data set naming standards, and, if done properly, will execute a VS1 or SVS jobstream with no changes.
On the 'con' side, you have the fact that it is system dependent for one thing. It is possible that changes to MVS could cause you to recode your exit. Further, you need a system programmer to implement and maintain it. What if this system programmer quits suddenly, just when a change to the exit is needed? This may not be a factor if you have 25 system programmers, but what if you have only one?
Finally, there are certain AMS command which ABSOLUTELY require a STEPCAT to function, even in MVS. What are you going to do if an application programmer needs to execute one of these commands in the jobstream? Most answer this objection by stating that their exit examines other parameters such as accounting fields before it disallows JOBCAT/STEPCAT. Of course the retort is once you let the cat out of the bag (give the application programmer the proper accounting code), it can obviously be used where it was not intended.

Protection is a kind of personal issue. Even if you use RACF, in the end it gets down to human beings controlling the computer, so you will have to decide this one yourself. I favor the first idea (having a control group) with some modifications to allow easy testing.

- Inform everyone about the JOBCAT/STEPCAT problem. Trying to hide it only causes more trouble later.

- Allocate some packs for the exclusive use of program testing, with a separate UCAT for those volumes.

- Allow JOBCAT/STEPCAT to the UCAT for the test volumes.

- Set up and enforce stiff penalties for testing using other catalogs.

The ISAM to VSAM Conversion Problem.

This is a relatively simple case at this point. In fact, it has already been covered, but it is such a frequnt occurance that I thought it deserved special mention.

- An ISAM data set is cataloged in a CVOL with the name A.B.C, with A being an alias to the CVOL in MCAT.
- The ISAM data set is converted to VSAM in order to use the ISAM interface. During testing, the name should remain the same in case the test fails.

The problem is that A.B.C will have to be placed in a VSAM catalog, but the alias in MCAT of 'A' is for a CVOL. Presumably, you can't change the alias because there are other data sets with the high level qualifier of 'A'.
During testing, this forces you to use JOBCAT/STEPCAT, or change the name of the cluster.
If you use JOBCAT/STEPCAT, you face the DISP=(,CATLG) problem if you allow that in the testing jobstreams.
If you change the cluster name, you must change the JCL to agree with it.

However distasteful, most customers end up changing the name.

Review of Key Catalog Concepts.

1. Since the IMBED option is forced, the catalog is 1/3, 1/4, or 1/5 larger than you might expect. The choice of the fraction depends on the CA size.

2. Every catalog entry requires a separate entry is each keyrange. Thus, the minimum number of bytes used for a single logical entry is 512 + 47 = 559. In particular, this is also true for nonVSAM entries.

3. The LKR cannot be split because an entry in the LKR is always a full CI. The HKR can and does split.

4. All KSDS clusters used compressed generic keys to form index entries. This has the effect of 'targeting' a specific HKR entry to one, and only one, CI in the HKR. If it doesn't fit, a split in the HKR will occur. If the split cannot be done, the catalog request will be rejected, regardless of space availability elsewhere in the catalog. Thus the catalog can appear 'full' when in fact it isn't.

5. There is no reorganization utility that will reclaim unused space in a catalog. The catalog must be rebuilt in this case.

6. The MCAT in MVS is required to hold the entries for the page and swap data sets, which are VSAM data sets. This forces VSAM volume ownership to those volumes. This implies that ALL VSAM entities on those particular volumes must be cataloged in MCAT.

7. If your choice of data set naming conventions causes entries in a VSAM catalog that are always in sequence (ascending or descending), the generic structure of the compressed keys in the index will not reuse space emptied by deleted entries.
   It the severest case, the whole catalog can be empty except for 10 entries in the last HKR CI, yet VSAM will try to place a sequential entry in that last HKR.

8. The use of GDGs in MVS catalogs, where the GDGs are set up to continually increment the generation number, use the catalog space as described in item #7 above.
   GDGs that are set up to reset back to generation one at some point do reuse the space when the reset occurs, but if you keep only the last 7 out of 365 generations, it will take a long time to finally reuse the space.

9. The use of the CATALOG parameter in AMS commands is known as forcing a 'directed' catalog search and will override any search strategy in any system.

10. Use of JOBCAT/STEPCAT is required in VS1 and SVS to use UCATs; it is optional in MVS due to the fact that MVS supports the use of alias entries in the MCAT to direct the search of UCATs.
    If JOBCAT/STEPCAT is used in any system, it is also interpreted as a 'direct' or 'directed' search.

11. If JOBCAT/STEPCAT is used in MVS, it will not allow catalog management to search MCAT for a possible alias to another catalog, WHEN PUTTING AN ENTRY INTO THE CATALOG.
    It will search MCAT for possible alias entries WHEN TRYING TO LOCATE AN ENTRY.

12. Item #11 becomes most serious when you consider that coding a DISP=(,CATLG) parameter on your DD statements obeys the rule laid down in item #11.
    This produces the fact that a jobsteam using JOBCAT/STEPCAT in VS1 or SVS can cause different results in MVS.

13. CVOLs cannot be named in JOBCAT/STEPCAT statements.
This means that the only way you can direct an entry to a CVOL is to name it correctly, use IEHPROGM, or code your own program incorporating CAMLIST macro support.
This means data set naming conventions become vital in MVS.

14. You cannot make MCAT a 'temporary' UCAT by naming it in a JOBCAT/STEPCAT. If you try, you do not get an error; the DD statement naming the MCAT (it could be a concatenation) is ignored for the purposes of catalog search strategy. (It is not ignored by allocation.)

15. Cataloging a new generation of a GDG forms one of the major exceptions to the rules above. In this case, the catalog management routines WILL search MCAT for alias entries EVEN IF JOBCAT or STEPCAT statements are provided.

16. When a JOBCAT or STEPCAT is used, a PCCB is used in the address space to indicate it's presence. The PCCB can contain a maximum of two names - the true name of the catalog and one alias. If your data set naming conventions utilize one of those two names, the catalog search will end in your PCCB chain; if some other name is utilized, MCAT will be searched for additional alias entries for the catalog that you already have allocated via your PCCB.
For performance, this means that you should use a naming convention that uses the same high level qualifier in many data set names, and, that qualifier should be the one utilized to FIRST utilize the UCAT or CVOL.

17. If dynamic allocation is used in MVS instead of the JOBCAT/STEPCAT, the rules stated in item # 16 are the same.
Items # 16 and # 17 probably affect the TSO user most. An alias of the TSO userid greatly helps the catalog search time.

18. A CVOL is allocated to an address space using PCCB logic just like a UCAT. Therefore, rules governing it's usage are identical to those for UCATs with the exception that CVOLs cannot be named in JOBCAT/STEPCAT statements.

19. Certain catalog operations require the job to reach end of step before their presence will be detected by others; deleting and redefining an alias is one example. (This has to do with the timing of the CLOSE issued by catalog management to a catalog.)
Since a TSO session is viewed as one step, a TSO user will have to LOGOFF and then LOGON again if these types of requests are to be immediately known to all other users of the system.

20. If a UCAT has a single level data set name, that name will itself act like an alias in MCAT and data sets with the exact same first level qualifier will be directed to that catalog.
If the UCAT has a qualified name itself, it's first level qualifier will NOT act like an alias in MCAT. You will have to physically issue the DEFINE ALIAS command and place alias entries into MCAT yourself.
(I am not recommending single level names for anything. It is best to follow a single naming convention consisting of qualified names for all, rather than grant this type of exception.)

What Can Go Wrong In A Catalog.

Most of the problems surrounding this area revolve around the fact that many people continue to view VSAM catalogs as a sort of extension to CVOL's, which were much less complicated internally. To answer the question of catalog errors, one must first understand some basic facts about catalog processing.

The designers of VSAM wanted catalogs to be inherently sharable, so they did away with the VSAM share options with respect to catalogs. That is why all catalogs show the share options as SHR(3,3) in a listing, even though all catalogs are able to be shared among multiple systems. The basic implementation of VSAM catalog sharing works as follows:

- When the first VSAM catalog is opened in a system, a CAXWA is established in global storage. (CAXWA - Catalog Auxiliary Work Area.) The CAXWA contains the number of users of a catalog, the equivalent of the CCR, the location of the catalog's ACB, the location of RPL's used to handle requests against this catalog, and information about the CRA if the catalog is recoverable.
- The user that opened the catalog will receive a PCCB (Private Catalog Control Block) chained to the active JSCB in the private area (partition, region, or address space).
- If additional users open the same catalog, the individual users each get a PCCB chained to their active JSCB, but the CAXWA chain is not modified; instead, the count of the number of users in the CAXWA is incremented.
- If a new catalog is opened, a new CAXWA is added to the CAXWA chain.
- Whenever a 'new' catalog is specified by a user, the chain of existing CAXWAs is always searched first. If an existing CAXWA is located, only the PCCB is built in the user's private area. In the case of JOBCAT/STEPCAT, the user's PCCBs determine the order of catalog search.
- Special processing occurs if the user specifies the master catalog as the 'new' catalog. Since the name of the catalog is in the CAXWA, VSAM determines that it is the master and NO ADDITIONAL PCCB IS GENERATED IN THE PRIVATE AREA. This implies you cannot override the order of searching the master catalog by specifying it on a JOBCAT or STEPCAT.

Since all requests for a specific catalog funnel through the same CAXWA, VSAM can maintain read and write integrity on a single CPU. To handle the case of multiple CPUs sharing a catalog, some special processing is implemented:

- Device RESERVE/DEQ is automatic.
- All requests are handled in DIRECT mode, even if sequential access is required. (This accounts for the relatively slow processing speed of LISTCAT.)
- Buffer refresh is forced (like SHR(4,4)), but CI/CA splits are allowed (unlike SHR(4,4)).
- A VERIFY is issued when the catalog is opened.
- The CCR contains RBA information which is used to update ARDBs (Address Range Definition Blocks) in the case of HURBA (high used RBA) changes.
- A special EOV routine allows EDBs (Extent Definition Blocks) to be rebuilt in the case of extent changes.
- If any request returns 'invalid RBA', VERIFY is issued to update the ARDBs.
- The 'open for output' bit is not used for catalogs.

We are now in a position to discover what actions can cause catalog problems. I am going to assume from this point on that you have backed up your catalog along the lines discussed elsewhere in this handout, but I will explain some very special tools available to you in this section should your recovery turn out to be bad also.

AT NO TIME SHOULD YOU EVER HOPE TO FIX A CATALOG PROBLEM WITHOUT BACKUP!

The Multi-CPU Problem.

If you place a VSAM catalog on a shared device, VSAM recognizes this by inspection of the flags in the UCB and issues the necessary RESERVE/DEQ macros. If one system in a multi-CPU environment failed to generate that UCB as shared, VSAM would fail to issue the required RESERVE/DEQ macros on that CPU. Even though every machine still uses forced buffer refresh, the refresh on the CPU that does not have the shared UCB is not synchronized properly, so the same free CI can end up being allocated to two different entries because the free CI chain is controlled by the CCR, which is now not being handled properly by the CPU without that shared UCB. The implecations of this problem are:

- The problem can go unnoticed for long periods of time if all catalog operations take place one one CPU. (While only one CPU theoretically modifies the catalog, secondary allocation is dynamic for VSAM, and if it occurs on the badly designed CPU, you wind up with an 'intermittent' catalog failure problem.)
- It seems obvious that any system changes done to one CPU (like an I/O gen.) will have to be done to all systems at the same time.
- Maintenance must be applied to all systems at the same time.
- VM users should treat their virtual machines as operating in a shared CPU environment for purposes of VSAM catalog sharing.

The Maintenance Problem.

VSAM PTF and APAR corrections can change the interpretation of catalog fields, notably the CCR. While this is usually documentated in the maintenance, is is usually NOT documented in the VSAM SRLs or PLMs until the next release of the manual. This means that the people involved in applying maintenance need to be aware that VSAM maintenance may affect catalog operations. Either the catalog should be rebuilt after major VSAM maintenance application (like the refresh release 7808), or great care should be exercised to ascertain whether catalog internal operations have been changed. The use of the CCR free CI fields was changed years ago by preformatting all free CIs as explained on page 6, but users still find catalogs that were built prior to that time in daily use. This can result in the catalog appearing to to out of free CIs prematurely; this may be only an annoyance, but anything that makes catalog operations more difficult should be corrected.

The I/O Interruption problem.

To understand this problem, you should have some idea of the basic relations that exist in a catalog for at least a simple KSDS. Let's list the relationships, omitting timestamp considerations:

- A total of 6 entries will exist for the KSDS - 3 LKR CIs for the cluster, data, and index components, and 3 HKR entries which tie the HKR to the individual LKR entries.
- The cluster record will contain an association field which ties it to each component and each component will have a similiar field associating it with its cluster.
- The volume record will have its space map set of fields updated. Further, the volume record has a special 'Data Set Directory' set of fields which associates the volume record with the data sets on that volume.
- The CRA must have duplicate records for entries on its volume. These entries are tied back to the catalog instead of having a separate chain of associations within the CRA. However, the volume record in the CRA describes space only on that volume and is not tied directly back to the catalog volume record.

It should be obvious that many I/O operations will be necessay to DEFINE a simple KSDS cluster.

For the Volume records, we must:

Update the volume record in the CRA for the data component.
Update the volume record in the catalog for the data component.
Update the volume record in the CRA for the index component.
Update the volume record in the catalog for the index component.

For the Data component, we must:

Write the data record in the CRA.
Write the data record in the catalog.
Write the true name entry in the catalog's HKR.
Update the catalog's CCR to reflect the usage of a free CI.

For the Cluster component, we must:

Do the exact same operations as described under the Data component.

For the Index component, we must:

Do the exact same operations as described under the Data component.

Recalling that an update requires two I/O operations, we get a total of 23 I/O requests to complete the DEFINE. It would seem obvious that unless all 23 operations were complete, we would have an entry that is only partially valid. Some key considerations are:

- VSAM will always check all associations in the catalog before attempting to process a cluster. This checking is performed regardless of the type of processing that is going to be performed. This means that VSAM WILL NOT DELETE AN ENTRY WHICH FAILS ANY ASSOCIATION CHECK!
- VSAM does NOT validate the entries in the CRA every time the cluster is opened. This means it is possible to have the CRA out of synch with the corresponding catalog records and not know it. This usually happens when someone restores a downlevel volume with IEHDASDR and uses SUPERZAP to fix timestamp mismatches instead of using RESETCAT.
- If an operator cancels an AMS job in the midst of a DEFINE or DELETE operation, the partial entry situation can occur. Note that this is true whether the operator used just plain CANCEL or CANCEL FORCE in MVS.
- There is no AMS command which will 'fix' a partial entry.
- If you continue to use a catalog which has partial entries in it, various 'funny' things can occur depending on when the I/O sequence was interrupted. This specifically involves the CCR record, i.e., two HKR entries can show the same LKR entry if the interruption occurs just prior to updating a CCR.

The first question to answer is the one about setting up some sort of ESTAE or STAE environment that will back out the damaged chains in the event of an operator cancel. First, you must remember that only MVS allows interception of an operator cancel, and VSAM is supposed to be compatible at least across OS/VS systems. But assuming that problem could be solved, the more serious case is that the entire catalog processing is done in global storage. Since backout usually involves keeping multiple records in storage until the I/O is validated, significant increases in the amount of global storage tied up for a catalog operation would result. Further, these operations can result in mounts and/or staging operations on the MSS unit, so OTHER USERS OF THE CATALOG MIGHT BE PREVENTED FROM ACCESSING IT FOR SIGNIFICANT TIME PERIODS.

It should be noted that there is STAE/ESTAE processing in catalog management today, but it does not solve all the problems mentioned above. IBM is investigating ways of improving the present facility, but as you can now see, the problem is much more complicated that first imagined.

The next question in this area involves making AMS noncancellable. (This is done by making an entry in the PPT - Program Properties Table - in the nucleus.) The problem here is that catalog errors can cause AMS commands to go into loops. Specifically, LISTCAT can go into a loop if certain catalog pointers are inaccurate. Further, while an MVS user can cancel a noncancellable job by using the FORCE option of cancel (other users will have to re-IPL), all of these techniques can introduce more catalog errors than already exist, for VSAM never flushes any buffers on an abend much less a force cancel or re-IPL!
Note that most of the problems involved with the interruption of the I/O do not damage the catalog to the extent it can no longer function; most of the errors involve not being able to DELETE a partial entry which then prevents you from using that particular name until the catalog is repaired. What is needed then is some sort of utility or utilities that will:

1. Identify the catalog problems.
2. Fix the catalog by overlaying records in the catalog.

I WANT TO EMPHASIZE A SECOND TIME THAT USE OF THE UTILITY TO OVERLAY CATALOG RECORDS THAT I WILL DESCRIBE SHORTLY IS IN ITSELF VERY TIME CONSUMING BECAUSE YOU MUST KNOW EXACTLY, DOWN TO THE BIT LEVEL, WHAT MUST BE CHANGED. THUS YOU SHOULD ALWAYS USE CATALOG BACKUP AND RECOVERY TECHNIQUES INSTEAD OF THIS METHOD IF AT ALL POSSIBLE.

Are there other ways a catalog can be damaged? Yes. Take a look at the next section.

The Catalog ENQ/DEQ Problem.

As described earlier, there are several ENQ macros executed by catalog management to control updates to the catalog and also to guarantee the integrity of the all important CAXWA and PCCB chains. If you backup a catalog in the midst of some catalog update, the backup is obviously potentially damaged depending on the exact sequence of catalog write operations.

- The REPRO command does not do ENQ on catalog resources.
- IEHDASDR does not do ENQ on catalog resources.
- The ALTER REMOVEVOLUME command does not check to see if the controlling catalog is OPEN.

Since REPRO and IEHDASDR do not ENQ on catalog resources, it is possible for either of them to be in execution while catalog operations are still not complete for a DEFINE, DELETE, ALTER, etc. This can result in the partial entry described above, even though all I/O operations complete successfully.

The ALTER REMOVEVOLUME situation means that the CAXWA chain can still state that a volume with a catalog on it exists when in fact it does not. This in turn leads to 'catalog error' when a second user (esp. in a TSO system) tries to access the now nonexistent catalog.

Note that both of these situations are not errors in the VSAM code as the REPRO is a data set utility, not a catalog utility, so it opens the catalog as a data set, not as a catalog. The ALTER RVOL command was specifically designed NOT to check for catalog activity as it is presumed that you are trying to clean up a volume for which no catalog exists. Checking of VSAM controls in this case would prevent ALTER RVOL from doing what it is supposed to do normally.
IEHDASDR is not a VSAM utility, so there is no reason for it to check VSAM resources.

There are only two known methods for absolutely guaranteeing that there are no users of a catalog:

- Re-IPL the system.
- Write a program to search the CAXWA chain.

Finding Out What Is Wrong With Your Catalog.

There are several aids which you can use to track catalog activity - keep an eye on it, so to speak. The main idea here is that the best cure for catalog ills is preventive medicine. Here is a laundry list of aids that can be used to moniter catalog activity:

LISTCAT — If it loops, skips a part of the catalog, or prints error messages, suspect a damaged catalog.
Also, watch the high used RBA field for the catalog data component. If it gets near the high allocated RBA, catalog full could be just around the corner.

PRINT — This AMS command reads the catalog as a data set. You can use the FROMKEY option to force it to print the HKR, or portions of the HKR, and either FROMKEY or FROMADDR to print selected portions of the LKR.
Use SKIP(3)COUNT(1) to print the CCR. No capability to print the CRA however.

IEHDASDR — Use the print option to examine the catalogs SDR to locate catalog extents, passwords, etc. This will also print the CRA.

LISTCRA — Compares CRA records with the corresponding catalog entries.

LOGREC — Check for hardware problems on catalog volume. In MVS, can also check for software problems in catalog management, VSAM logic, etc. routines.

SMF — Often the only source of just what was DEFINed, DELETEd, etc. All catalog updates are logged in SMF.

TRAP — A catalog trap exists to trap catalog return codes, etc. It is described in the Catalog M'g't. PLM for MVS and in the VSAM Logic PLM for VS1 and SVS. Look under 'Debug Aids'.
Note that these traps are normally used in conjunction with GTF to diagnose really difficult problems.

USER PGM — The user can write programs that access the catalog as a data set. The requirements are listed in Options for Advanced Applications. In particular, MVS requires such programs to be authorized.

IDACATCK — Neat program available only through your IBM PSR. This probram is the property of of IBM and is not considered part of the AMS utilities. Your IBM SE can assist you in using this program. IDACATCK:

Performs the following checks.

- Free chain.
- Dead record. (LKR record not on any chain.)
- HKR relationship to LKR.
- LKR associations.

Gives you the following statistics.

- Number of formatted records.
- Number of records used in the HKR.
- Number of entries of each type in the catalog.
- Number of 'unavailable' records - this amounts to the total number of records available for HKR usage.

For those customers wishing to track the number of records available in the HKR, running IDACATCK for its statistics alone is a worthwhile investment. However, even IDACATCK will not tell you if certain ranges in the HKR are full, i.e., diagnose the fact that GDGs are filling up certain ranges in the catalog. If your catalog is small, you might just be able to get by via IEHDASDR prints of the HKR and use simple inspection of the printout. Anything more than that will require a user program.

IDACATCK will give you numerous error codes if it finds errors, and it will dump the records in error. This means you can use the output of IDACATCK to fix the catalog IF, and only IF, you know how to interpret a catalog record AND know what is SHOULD look like.

THIS IS MY THIRD WARNING. THE NEXT UTILITY DESCRIBED WILL ALLOW YOU TO FIX A BAD CATALOG RECORD. THIS UTILITY DOES NOT, AND IS NOT INTENDED, TO REPLACE NORMAL BACKUP AND RECOVERY PROCEDURES. DETERMINING WHAT CAUSED AN ERROR AND WHAT IT HAS DONE TO A CATALOG CAN CONSUME MORE TIME THAN REBUILDING THE CATALOG FROM SCRATCH.

The utility IDACATFX is a utility that is available through your IBM PSR and can be used to overlay catalog records. This utility is the property of IBM and does not form part of the AMS component of VSAM. There is no documentation in the form of manuals for either IDACATCK or IDACATFX; the programs are self-documenting. Seek the assistance of your IBM SE in using these utilities, i.e., IDACATCK and IDACATFX.

IDACATFX will allow you to do the following:

- Display or Print any part of a catalog by RBA, CI number, or data set name.
- Add records to the HKR.
- Delete records in the HKR or LKR. If the deleted record is in the LKR, it is put on the free chain for you.
- A ZAP function by CI number or RBA. (It is almost the same as SUPERZAP, i.e., it uses the VER/REP logic.
- A free chain rebuild, which will collect all records that are free or dead and put them back on the free chain.

Note:
A third utility called VSAMDIAG exists in the same manner as IDACATCK and IDACATFX, i.e., it is IBM property and available only through your IBM PSR. VSAMDIAG does checking of KSDS clusters, i.e., it checks each CI in a KSDS for validity. During this check, it prints statistics as to the exact amount of freespace available. While it was not designed for catalogs, and to my knowledge has not been tested against catalogs, it may be of some use to you in establishing the amount of freespace in your catalog.

Other useful catalog information.

1. One way to monitor the catalog full situation is to define a catalog with secondary space and then watch for the secondary space to be allocated. While this should always be done to avoid a catalog full condition, users complain that performance falls off as secondary extents are allocated. However, if you suballocate a catalog, you can define secondary space for the catalog itself, but omit it for the VSAM space itself.

2. The space map in the catalog is device dependent. This is true even for 3330 model 1 and 3330 model 11. This means that you cannot use IEHDASDR to 'copy' a catalog from a 3330 model 1 to a 3330 model 11, because the space map will show only half of the volume. (The space map is always built to reflect the TOTAL number of tracks on a volume, not just the tracks available to VSAM.)

3. The REPRO command of AMS will unload and reload catalogs, but during the reload, it reorganizes the catalog only if the target catalog is empty. (Read 'reorganize' as rebuild the free chain.) If you reload into a non-empty catalog, no rebuilding of the free chain takes place, so any free chain errors already in the target will stay there.

4. The REPRO copycat function of MVS ends up by defining the source catalog as a normal KSDS in the target catalog. If you fail to complete the procedure outlined in the AMS manual (which involves deleting this KSDS) you wind up with two catalogs owning the same space. If you now were to execute a DELETE UCAT FORCE against the target catalog (assuming you decided you did something wrong and wanted to start over), the delete would delete both catalogs and you wind up with nothing!

5. MVS formats a new catalog on the first update to it. If the catalog is large, this preformatting can take a long time. You can use a time of 30 seconds per cylinder on a 3330 as a first approx.

6. The catalog contains the location of the FMT1 DSCBs that define VSAM space. If you use a program to reorganize your pack, making sure it doesn't physically move the actual VSAM space is not enough. You have to make sure it doesn't reorganize the VTOC as well!

7. Remember that there is no dynamic deallocation of a catalog. In MVS, this means that if a TSO user allocates a catalog, it will remain allocated until logoff. It also means that if a TSO user defines a new catalog, it will not be usable to others until logoff because VSAM CLOSE processing updates the statistics that indicate the new catalog can be processed. (Until this is done, various pointers, such as the high used RBA, are still zero.) This explains why a second user can 'see' the new catalog in the master catalog's type U entries, yet cannot process it.

VSAM Catalog Performance.

By this time you should realize that you can get increased performance of any VSAM KSDS by getting more buffers allocated to the right functions. Since a catalog is processed using direct techniques at all times, this means that you need to get catalog management to allocate more buffers for the catalog index records. However, there is no direct way of doing this. Let's examine the way catalog management assigns buffers.

First, you specify the amount of bufferspace to be used in total on the BUFFERSPACE parameter in the DEFINE for the catalog. BUFFERSPACE defaults to 3072. If you take the default, VSAM will do the following:

Two RPLs will be set up. This allows two concurrent catalog requests. Each of these RPLs will be associated with one index and one data buffer. Since all catalog CIs are 512 bytes, this uses up 2048 bytes.
One index buffer will be set aside for the highest level index record.
One more buffer is set aside for a CI split work area. Total is 3072 bytes.

The AMS manual states the maximum you can specify for BUFFERSPACE is 8192. (This is incorrect.)
If you code 8192 in BUFFERSPACE, VSAM will still set aside the one buffer for the highest level index and the one for the CI split work area. This would leave 7K.
However, since each catalog record is only 512 bytes, we can get 14 buffers out of the 7K.
VSAM will use those 14 buffers to support 7 RPLs, where each RPL uses a pair of buffers as described above.

The reason the AMS manual lists 8192 as the maximum for BUFFERSPACE is that VSAM will NEVER allocate more than 7 RPLs, but if BUFFERSPACE is larger than 8192, it will use the extra space for index buffers. If you specify 9216, 2 extra index buffers will be available to handle the index set. This will greatly improve catalog performance, because the only buffer available without this specification for index set records is the high level index buffer. Thus the high level index must be overlayed without this specification.
You can specify a value greater than 9216, but since most catalogs have only a few index set records, it is doubtful that going much above 9K will yield even greater throughput.

WARNING:
You should implement this only for NONSHARED catalogs. The reason is that for shared catalogs, all buffers are refreshed as required. A large BUFFERSPACE value will aggravate the buffer refresh times.

The easiest problem that you can 'see' without much thought is the case of a VSAM catalog shared between two CPUs where the user has failed to define the proper UCBs as shared (or the operator mounts the pack on a unit that is set up deliberately nonshared). Obviously, VSAM does not execute any device RESERVE, so it is possible to have different HKR entries pointing to the same LKR CI, because the CAXWA in each machine contains information relating to the position of the next free LKR CI. Even though VSAM forces buffer refresh for catalogs, it doesn't have to do a buffer refresh to take multiple free LKR CIs off the chain, as in the case of allocation of a VSAM cluster. Thus, if both machines happen to DEFINE a cluster at the same time, both can refresh the CCR, get the same result, and allocate the same free CIs.

SHARING IN OS/VS VSAM

The following material assumes you are NOT going to use:

GSR	Global Shered Resources
LSR	Local Shared Resources
CBIC	Control Blocks in Common
ICI	Improved Control Interval processing.

However, a general knowledge of catalog structure is assumed for the portion covering catalog sharing in single and multiple CPU environments.

Shareoptions Support.

While all current COBOL and PL/1 compiliers support the latest version of VSAM (enhanced VSAM), there are still some VSAM options which can be coded by the user only in Assembler. One example is an MVS option called CBIC - Control Blocks In Common. Some IBM products like CICS, IMS, and VSPC use VSAM options that would normally be available only in Assembler, but as of this writing (June, 1979), none of these products offer any MORE sharing facilities than would be available using normal VSAM.
It is important that you understand that just because you use IMS or CICS, you are NOT absolved from thinking about VSAM sharing if you intend to share the data bases with other programs.

Preliminaries.

Why are VSAM statistics important? VSAM keeps statistics in the AMDSB - Access Method Data Statistics Block - which you will find in the data and index catalog record. In the AMDSB you will also find, at offset x'1C', the location of the highest level index record. When a VSAM cluster is opened, the AMDSB is brought into storage and chained to it you will find:

- The first ARDB - Address Range Definition Definition Block - which, together with additional ARDBs as required, sets up the range of allowable keys (implements the VSAM KRNG parameter) along with the RBA values that control a particular range.

Open will cause another set of control blocks to be built as well:

- The EDBs - Extent Definition Blocks - which relate DASD extents to RBAs.
- The LPMB - Logical to Physical Mapping Blocks - which contain information about how to convert VSAM RBAs to physical track addresses, i.e., MBBCCHHR.

Now all of these control blocks are built in your partition, region, or address space (hereafter P,R, or AS) and there is also a separate set for the index and data components. From this information, three critical points emerge:

1. VSAM does not broadcast control block changes from one P, R, or AS to the others. Whatever is done in one P, R, or AS is NOT reflected in the others.
2. The key information for building the blocks is housed in the catalog. Unless the catalog records are accurate, you cannot expect to process a VSAM cluster correctly.
3. The basic time that the catalog records are updated is CLOSE. There are exceptions, but a cardinal rule will be that if a VSAM cluster is not properly closed, the integrity of that cluster is in doubt.

Let's examine the AMDSB more closely. When a VSAM KSDS is loaded, the index CIs are sequentially built. Thus, the first record in the index will be the sequence set for the first CA. When the second CA is loaded, a second sequence set record is put into the index. However, at this point VSAM realizes that a higher level index record is needed, so it writes the index set record in the third CI in the index. As more CAs are loaded, more sequence set records will be written in the index. When the time comes to write a second index set record, VSAM will build a third level index record and write it in the next available CI in the index. (The basic rule is that if two index set records exist at a given level, a higher level index set record must be built, so that the highest level in any index will be a single CI.) Now the AMDSB records the position of that highest level CI. If the AMDSB is not updated properly, or if some older version of a catalog is used to process a VSAM cluster, whole sections of the cluster may suddenly be nonprocessable. How can these things happen?

Well, suppose you decide to concurrently insert records from two or more P, R, or AS and you use no control whatsoever. Now all users would start with the same copy of the control blocks because each

copy is built from the same catalog records. As CI and CA splits are done by the various users, their individual control blocks will be updated, but their updates to those control blocks will not be reflected in any other P, R, or AS. Note in particular that if a CA split has occurred in one P, R, or AS, it will not be reflected in others. This has become known as the read integrity issue, because records in the CA that was split can suddenly appear to be unable to be reached due to the fact that the ARDBs in all P, R, and AS have not been updated.
It is obvious that if the catalog records are updated by close, then the order of issuing the CLOSE macros is important. However, there are two important exceptions to the rule that close updates the catalog. These exceptions are:

1. The catalog is always immediately updated if additional space is allocated to the cluster.
2. The catalog is always updated immediately if the highest level index set record is split.

At this point, you might be a little confused between the catalog and the actual data in the cluster. Remember, VSAM updates the cluster all right, but you can still get a no record found condition after someone has done a simple CI split. If you assume that the user that did the CI split has written all required CIs back to the data set, then you should be able to retrieve the records he just inserted, right? Not so. Remember that thing called buffer lookaside? It states that VSAM will use buffers associated with a string if they are flagged as valid. If you just happened to have that sequence set record in storage along with the other guy, your copy still reflects the old CI numbers. When you go to retrieve the record, VSAM will use the old sequence set information, retrieve the 'wrong' CI, and declare a no record found condition.
My point here is that too many users feel that as long as just one user does all the updates, all other users can read the cluster without implementing any controls at all and expect to be able to retrieve every record that was there before a CI or CA split.

For all requests, VSAM checks the RBA against the control blocks described on page 1, and if it's higher than the high used RBA, the request is failed with a feedback code of x'20'. Thus CA splits are more of a problem than CI splits (except a CI split in the last CA) because they cause numerous records to be moved beyond your current high used RBA. (Later, you will see that there are ways to solve this problem without resorting to reopening the cluster.) Secondary allocation is much more difficult because you need the additional EDBs that can be built only from catalog records. In general, if secondary allocation has occurred, you will need to reopen the cluster.

You might think you have a way to solve all these problems - the VERIFY macro. However, the VERIFY macro updates only the control blocks it sees in your P, R, or AS; it does not build additional control blocks to represent the dynamically acquired extents by others.

At this juncture, you may say: "Well, just CLOSE and reOPEN. That will rebuild everything."
True. But, and this is a big but,
HOW DO YOU KNOW WHEN TO DO IT?
Here are some other interesting questions to ponder while you are at it.

1. How much reprogramming is required even if you are positive you know 'when'? Are you sure you have covered every case? Have you informed every programmer?
2. Can I recognize all necessary feedback codes in my language? COBOL does not return every individual feedback code to the user. Will an Assembler interface be necessary?
3. Are you aware that some IBM products, notably IMS, open all clusters as output clusters, whether they are going to be used for output or not? How will that affect your intended sharing?

Rather than try to answer these questions now, let me go on with the introductory material.

Why doesn't VSAM update my AMDSB and ARDB, EDB, and all the others, at least as an option?

This argument has come up again and again - anyone looking at VSAM sharing eventually asks the question. Two facts contribute to the argument:

1. OS/VS updates ISAM DCBs by maintaining a list of them in SQA.
2. DOS/VS, using share option 4, maintains 'global' information about a particular ACB by essentially putting the ACB and related CBs in it's GETVIZ area - like our PLPA.

Let's look at these arguments.

ISAM never allowed sequential insert capability and further, in the case of data set extension, you were forced to use DISP=MOD which used exclusive control by the initiator to prevent simultaneous anything! VSAM allows sequential insert. Now sequential insert implies deferred writing of buffers (as opposed to a direct update/insert which writes the buffers immediately). For VSAM to 'broadcast' the results of sequential inserts would mean VSAM would have to somehow quiesce ALL users of the cluster in ALL P, R or AS because those users could also be doing deferred writes.

Secondary allocation in VSAM involves not only changes to existing CBs like the ARDBs, but also creation of additional CBs - EDBs for example. ISAM never allowed any more blocks to be built - in fact, ISAM never allowed any secondary allocation at all!

In MVS, any communication with another address space involves SRB scheduling (or some equivalent function) due to the fact that no address space is addressable to any other address space. Further, MVS supports both MP and AP processors, so functions like disablement cannot be used to 'simplify' control block manipulation.

ISAM wasn't that good in the first place. If you were using in core indicies, ISAM did not do a buffer flush on inserts - that was left to the user. VSAM does force index buffer flush on inserts, and that helps. The point is, maybe you are doing more now with VSAM than you did with ISAM. That's why most of you never even knew that ISAM doesn't flush it's in core indicies.

There is no denying that DOS/VS will use a global concept in it's implimentation of share option 4, but the DOS SRLs plainly state that read integrity is not guaranteed - only write integrity. Before you begin screaming to get OS/VS to DOS/VS standards, how many times have you thought, BEFORE the application was brought up, that read intrgrity was not important, only to find AFTER implementation that read integrity IS VERY IMPORTANT?
I might add here that OS/VS does support a global option, but not as a share option. In OS/VS, it's called GSR (Global Shared Resources), but it involves much user coding. A new option in MVS called CBIC (Control Blocks In Common) is also available which again involves considerable coding on your part. These options, together with a third called LSR (Local Shared Resources) are implemented in certain IBM program products, notably IMS/VS and CICS/VS, although not all options are available in all products in every OS/VS system. For exact details, see your local IBM representative.

The VERIFY command vs. the VERIFY macro.

Since the AMS SRL indicates that the VERIFY command will update the catalog, most readers of the Options for Advanced Applications SRL (GC26-3819) assume the VERIFY macro will do the same - will not! The VERIFY command first OPENs the cluster, then issues the VERIFY macro, then CLOSEs the cluster. It's the CLOSE that updates the catalog, not the VERIFY macro. (The OPEN in the AMS command also ignores warning feedback codes; that's the reason the VERIFY command will 'fix'

cluster update timestamps and the 'opened for output' flags when the CLOSE is issued.) The VERIFY macro simply compares the content of your control blocks to what is supposed to be in the cluster. It does NOT generate additional control blocks for you, i.e., it will not 'find' additional extents.

The basic VERIFY macro processing consists of validating the location of the SEOFs - software end-of-files. Perhaps a review of what a SEOF looks like is in order. Consider the CIDF at the end of each CI. It has two fields - the location of the freespace and the amount of the freespace. A valid CIDF can be in one of only three states:

LOCATION	AMOUNT	
0	Max.	This CI is 100% freespace.
X	Y	This CI has some freespace.
RDF	0	This CI has no freespace.

Notice that the combination of all zeros is theoretically not possible. Now, since VSAM initializes all CIs to zeros when it writes a so called 'free' CI, a free CI will fall into the first case and its CIDF will NOT be zero. A SEOF is 'initialized' to binary zeros just like a free CI, but its CIDF is also zero!

Now let's take a look at where VSAM writes these SEOFs.

RRDS cluster. In the last CI. If every CI is now used, there is no SEOF.

ESDS cluster. In the last CI. If every CI is now used, there is no SEOF.

KSDS cluster. At the top of the last CA. If every CA is used, there is no SEOF. Note that the CIs after the high used RBA in the last CA are formatted as free CIs, not SEOFs.
If the cluster has key ranges, each range is treated separately.
The index, being a data set, is also treated separately.

The VERIFY macro processing consists is reading the theoretical SEOF for each key range and the index. If the CI read does not consist of a CI of all zeros, including the CIDF, VERIFY assumes the component has been extended and proceeds to find the new end of the cluster component by reading forward until it either finds a SEOF or comes to end-of-extent. Note that for RRDS and ESDS clusters this means reading each CI after the one that was supposed to contain the SEOF, but for KSDS clusters, only the first CI in each following CA need be read, since VSAM always formats a KSDS cluster one CA at a time. If VERIFY finds any discrepancy, your control blocks are updated, but of course these changes are not 'broadcast' to any other user. Remember that the VERIFY macro never updates the catalog - only CLOSE and EOV processing does that. The VERIFY command of AMS can update the catalog (timestamps, SEOF location) because it issues OPEN and CLOSE. VERIFY terminates immediately if the high used RBA is zero!*

By this time you should be painfully aware that if you wish to engage in concurrent update/insert activity, you are going to have to do something about secondary allocation, because unless you agree to CLOSE and reOPEN your cluster, there is no way for you to find out about secondary allocations caused by other users. The VERIFY macro will not find them. The problem with CLOSE and reOPEN, as stated earlier, is to determine WHEN to do them!
This leads to the problem of finding out when secondary allocation has occurred. You can find out that YOU caused secondary allocation by using either the journaling exit or by checking for feedback code x'04' after a x'00' is returned in register 15 for a PUT request. But there is no macro, code, or exit that will inform you that some other P, R, or AS caused secondary allocation. As stated previously, the only way to find out if another P, R, or AS caused secondary allocation would be to either build the logic into your code (probably via ENQ/DEQ or a user coded SVC), or to use global shared resources or CBIC. Either of which represents a major undertaking. For that reason, you might begin to think of not allowing secondary allocation for certain VSAM clusters at your installation.

* The high used RBA is zero during create mode. It is not set to a nonzero value until CLOSE. This means you can't VERIFY a cluster that 'bombed' during load!

VERIFY Command Processing in a Multi-CPU Environment.

The VERIFY command issues an OPEN for CNV processing in output mode. This means that to use it implies that the CLOSE will update the catalog. One of the fields updated is the 'opened-for-output' flag in the catalog. Now if other users are opened for output in a single CPU, the VSAM ENQ/DEQ logic will detect that and the opened-for-output flag will not be reset.
Further, many IBM products, such as IMS/VS, always use SHR(1,x) or SHR(2,x) and always open clusters for output regardless of the processing intended. In such an environment, the VERIFY command of AMS would be prevented from executing simultaneously due to the share options (implemented by the ENQs).

But, if you are running in a multi-CPU environment, the ENQs issued in one CPU are not known to other CPUs. Thus it is possible to run a command like VERIFY on one of the CPUs that has nothing running against the VSAM cluster in question in which case the opened-for-output flag in the catalog would be cleared. Of itself, this is not necessarily good or bad. Note that jobs that attempt to share the cluster for output on the first CPU will still be prevented from executing due to the ENQs done on that CPU. As long as all jobs terminate normally, no errors will be introduced.

However, suppose the job running on the first CPU terminates abnormally. Under normal circumstances later attempts to OPEN such clusters will return the feedback code x'74' warning the user that the cluster was not properly closed. But in this multi-CPU environment where a program running in another CPU has turned off the flag, the warning will not be issued and the user can start processing a cluster for output which has errors in its catalog records. Naturally, this can introduce more errors into the VSAM cluster; in fact, you should think of the VSAM cluster as permanently destroyed after such a situation and initiate recovery processing.

SAM ENQ/DEQ Processing.

To implement various share options, VSAM uses various combinations of names in the ENQ/DEQ macros. The convention is:

Major name: SYSVSAM
Minor name: CCCAAAAX where

- CCC — CI number of the catalog component record.
- AAAA — Address of the catalog's ACB.
- X — Has one of the following values:
  - I In use, i.e., input.
  - O In use, opened for output.
  - B Busy, i.e., in the process of being opened.

You should be aware of the following critical points:

- Separate ENQs are issued for each component of a cluster.
- An open for output will result in two ENQs, one using the 'I' and the other using the 'O', since a cluster opened for output can also do input functions.
- These ENQs are issued even if you code SHR(3,3) as they are used by catalog management to prevent someone from deleting your cluster while you are still processing it. (If we didn't do the ENQs for SHR(3,3), and someone deleted your cluster while you were still processing, you would write over 'empty' VSAM space. If that space was reallocated, you would be writing over another user's cluster!)
- As stated earlier, ENQ/DEQ provides no protection in a multi-CPU environment.

Now if you still intend to engage in the concurrent insert/update activity, you might want to limit a user to the space currently owned. Remember, even if you do that, a CA split can still occur in the current allocation. If you choose this route, remember also that you must be careful to specify no secondary allocation for the cluster; specifying no secondary allocation for VSAM space alone will NOT 'cancel' secondary allocation for a cluster as space will be dynamically allocated if required. The effect of this limit would be to eliminate the need for close and reopen because no new EDBs would ever be generated, but the VERIFY macro (or command, depending on the situation) would still be necessary to handle the CA splits in the current space. One easy method here might be to implement the VERIFY macro after a feedback code of x'10' on a retrieval - no record found - as part of your error recovery. (This x'10' might be due to the fact that your record was moved to a new CA during a CA split, but your ARDBs still reflect the old ranges.)
Note that the use of VERIFY requires CNV processing. If your ACB does not allow CNV processing, you would have to close and reopen it. Since the catalog is updated immediately on any change that involves the high used RBA, this close and reopen would be sufficient and the VERIFY would be pure overhead. If you anticipate many no record found conditions, either VERIFY with CNV processing or close and reopen logic will be expensive. Remember, VERIFY must verify every key range and the index component, so many I/Os may be required.

You could decide to eliminate CA splits as well, and then all of the overhead described above would be eliminated. One way to do that is to use SHR(4,x), but that option in itself is costly because it forces a buffer refresh on every direct retrieval, i.e., buffer lookaside is negated. Another approach is to initialize your cluster yourself and then do only updates to 'existing' records. Still another method might be to initialize your cluster and then delete every record. Since the index is not affected by mass deletions, you could insert records later and never cause any splits. This eliminates all thought of freespace however.

Every one of the techniques described so far has something distasteful; it is really up to you to decide just how far you want to go in VSAM sharing, but the net of it is that VSAM really doesn't provide much in the way of sharing. I think that you should design applications with the idea that VSAM provides nothing in the way of sharing facilities. Then, whatever meager crumbs you get will really look good to you (and be quite helpful). The worst cases always involve users who thought VSAM was going to provide everything.

End of Preliminaries.

Initiator Sharing. (DISP=OLD vs. DISP=SHR)

Since no VSAM cluster can be a temporary data set, the initiator is forced to perform data set name enqueue for all VSAM clusters according to whether DISP=OLD or DISP=SHR is coded. This is significant because it means there will always be a QCB/QEL for a VSAM cluster whether it is open or not. When the cluster is opened, more ENQs will be issued as described on page 5. If you code DISP=OLD, the initiator will guarantee that you are the sole user of that cluster on that CPU. This has the net effect of forcing SHR(1,3) and assuming you will open for output. You must code DISP=SHR for the VSAM share-options to take effect.

Note: DISP=OLD forces SHR(1,3) even if SHR(1,4) was coded. This can have drastic effects in a multi-CPU environment.

Note2: If your program name appears in the program properties table - (PPT) - initiator data set name enqueuing can be surpressed. This may allow more VSAM users to be online than you anticipated. The VSAM shareoptions can help you here, IF YOU CODED THE CORRECT ONES.

VSAM Cross Region Sharing. (SHAREOPTIONS)

First, a review of some important points. Note that the discussion that follows is centered around KSDS processing because it is the most difficult to control with regard to shareoptions. Users are advised to consider RRDS and ESDS clusters, or combinations of all three types to simplify shared processing if possible.

1. The AMDSB and the information to build ARDBs, EDBs, etc. resides in the catalog in the cluster's D and I catalog records. OPEN builds control blocks in your P, R, or AS from this catalog information; the catalog information itself is updated only by CLOSE, CLOSE TYPE=T, EOV, secondary allocation, splits that change the high used RBA, and splits that change the number of index levels in the index component.

2. If you open an ACB that has MACRF=(OUT,...) coded, an 'opened for output' flag will be set in the catalog at offset x'9D' in the respective component. Since this flag is a single bit, it cannot be used to determine the number of users opened for output. The QCB/QEL chain described earlier is used with ENQ (.....),RET=TEST to determine the status of current users. Note that ENQ/DEQ is valid only in a single CPU or TCMP, never in a shared CPU environment.

3. The VERIFY macro never updates the catalog; only the CLOSE, CLOSE TYPE=T, EOV, and the split processing described in point #1 above change catalog records. The implication is that the VERIFY macro cannot be used to 'search' the catalog in an attempt to locate additional extents allocated by other users.

4. A CA split may or may not cause a catalog update. If the high-used RBA is changed, or if the number of index levels is changed, the catalog is updated. Secondary space allocation also causes an immediate catalog update. But CA splits that do not modify these items, i.e., a CA split in space already allocated, not affecting the number of index levels or high-used RBA, will NOT cause a catalog update.

5. The ENDREQ macro, in non-create mode, invalidates data buffers only associated with a string. If updates occurred, the data buffers are written back first, and if the buffers are associated with multiple strings, ENDREQ will wait for all other processing to complete before invalidating the buffers. All buffers associated with the string are invalidated, not just the one associated with the RPL indicated in the ENDREQ; if a sequential request is then directed at the string, all buffers will be reprimed. The implication is that you cannot force VSAM to reread the sequence set by using ENDREQ.

SHAREOPTIONS(1,x)

This option means VSAM will permit any number of read only users OR a single output, NEVER a combination of the two. This guarantees that if you are the output user, you are the only user - period! This option differs from initiator sharing in that if DISP=OLD is specified, the initiator will prohibit the step from executing if there are other users; SHAREOPTIONS(1,x) will allow the step to begin execution since presumably you coded DISP=SHR instead of DISP=OLD. The advantage is that a user might be able to continue processing if you get a bad feedback code from OPEN by switching to a different processing mode. The testing done by OPEN is straightforward:

- Issue an ENQ (...),RET=TEST to determine if there are other users of this cluster.
- If there are no other users, OPEN is successful regardless of ACB parameters.
- If there are other users, check their QEL. If the name in the QEL ends in an 'O', a user is opened for output and the request fails regardless of ACB parameters.
- If the QEL name ends in an 'I', your request will be honored only if your ACB indicates input only.

Note: Suppressing data set sharing in the PPT does NOT affect the ENQs issued by VSAM.

SHAREOPTIONS(2,x)

This option permits any number of read only users AND one, and only one, output user. This option differs considerably from either SHR(1,x) or initiator sharing via DISP=OLD in that the cluster can be modified by the output user while others are reading it. This has become known as the 'read integrity issue', and I will discuss it at some length because it's processing idiosyncrasies have ramifications in SHR(3,x) and SHR(4,x). Implementation is similiar to that of SHR(1,x), i.e., the ENQ (...),RET=TEST is used to determine the final letter in the QEL name of other users, but you are stopped only if that name ends in an 'O' and you are also attempting to open for output.

If you are the output user, SHR(2,x) guarantees write integrity since there cannot be any other output users, but if you are the input user, you have some liabilities. In reading the following, ALWAYS ASSUME YOU ARE THE READ ONLY USER!

Case 1. Insert with no CI/CA split and no change of the high-used RBA.

Suppose some other user inserts a record into the CI you now have in your buffer. If you request that record, you will get a no record found feedback code because VSAM performs buffer lookaside at the string level. If you happened to have some other CI in memory, VSAM would have been forced to read the cluster and would have obviously found the record. (Note that no sequence set modification has taken place, so it doesn't matter if the new CI you read is within the same CA.) If you are doing sequential processing, there will of course be no indication of record not found because the sequence set is used only to locate the CI number to be processed; the keys in the records are NOT checked against the compressed keys in the sequence set record.
You can 'solve' this problem simply by issuing an ENDREQ macro after a no record found feedback code followed by another GET. If the insert has been completed, you will get the record. Note that you have no way of knowing that the other user has completed the insert; all you are doing here is eliminating any conflicts caused by VSAM buffer lookaside.

Case 2. Insert causing a CI split, but no high-used RBA change. (Not in the last CA.)

This case causes a change in the sequence set record, but no change in any catalog record. The problem here is that the ENDREQ will not invalidate the sequence set buffer, so using it will be of no value, as the reread will use the old sequence set information due to buffer lookaside.
Now there is no lookaside across strings in VSAM, so regardless of what other strings are doing, if you position your string to a different CA, and then reposition back to the original CA, VSAM will be forced to reread the sequence set. This suggests the possibility of using the POINT macro.
One problem here is that the user is not likely to know which CA is being processed, so it will be difficult to determine the target for the POINT macros. To solve that, you could load the first CA with junk records and always POINT to it and back, or you could process using BUFNI=1. If you have at least two CAs, BUFNI=1 combined with a direct GET will force the reread of the sequence set every time without any POINT, because a direct GET always searches the index top down. With only a single index buffer, VSAM would be forced to read the sequence set every time as the index set buffers would overlay it on each request! IMBED and REPLICATE would be a must in this situation, but even so, the performance implication here is drastic - a minimum of 2 extra I/Os for every read!
IN ANY EVENT, BE SURE TO DOCUMENT WHAT YOU ARE DOING SO THAT SUBSEQUENT RELOADS AND/OR JCL CHANGES TO NOT INVALIDATE YOUR WORK.

Case 3. CI split in the last CA or insert into the last CI but no secondary allocation. (High-used RBA change.

This case introduces a new problem in that your ARDB is now incorrect. Since VSAM checks your requests gainst your ARDBs (one ARDB for each key range), it is possible for a read only user to get a feedback code of x'20' - RBA not for any record in the cluster - even though the record actually exists.

You can use the VERIFY macro to 'solve' this case inasmuch as there has been no secondary allocation. (Recall that the VERIFY macro uses only the control blocks you have now to perform the verify; it does not read the catalog to find other extents.)

Be aware that CLOSE followed by OPEN may not be of much help here because as stated once before, you have no way of knowing that the insert has taken place. Since the catalog is updated on a high-used RBA change, a CLOSE followed by an OPEN would solve this problem, but at what price? Of course if the programmer suspects that this is an isolated case, i.e., the programmer is aware that processing is being done in the last CA and a CLOSE and reOPEN have not been done, then a CLOSE and reOPEN are in order. The VERIFY macro is much faster, BUT THE VERIFY MACRO REQUIRES THE USE OF CNV PROCESSING. If you are not already in CNV mode, you will have to CLOSE and reOPEN anyway! The net result of this discussion is that if you intend to 'solve' the read integrity issue, CNV processing can solve it a lot faster, but CNV processing is prohibited in high-level languages at this time.

If you are the user doing the insert, you don't have to worry about either multiple string processing in your ACB or the presence of other ACBs in your program. If you have multiple strings, VSAM provides full integrity in handling them; if you have multiple ACBs, the ENQ mechandism described earlier will catch errors in sharing a cluster within your program in the same way it catches errors between programs.

Case 4. CA split within the present allocation.

This is the same as Case 3 because there are no new EDBs, but the high-used RBA has changed.

Case 5. Any insert causing secondary allocation.

This case causes the catalog to be updated and requires a new EDB in your P, R, or AS for correct processing. The ARDB controlling the key range in question must also be changed as it contains:

ARDHKRBA	x'08'	RBA of the CI containing the highest key for this key range.
ARDHRBA	x'0C'	RBA of the next free CI at the end of the key range.
ARDERBA	x'10'	RBA of the highest CI allocated to the key range.

Other fields are updated also, but these are listed to convince you that attempting to process a cluster with these fields in error can lead to all sorts of unpredictable results.

Is is vital for you to realize that one half of an old CA can become permanently unaddressable by you because it is moved to a new CA which is not reflected in your control blocks. The VERIFY macro cannot help you here because it checks only what it sees via your control blocks, and yours don't have the new extent listed. At first glance, it would seem that a CLOSE followed by an OPEN would solve this case as well as all the previous ones. Before you make that assumption, read on.

A certain user added CLOSE and reOPEN logic to a VSAM program that used SHR(2,x). To the dismay of all, the problem of missing records persisted. After exhaustive checking (via GTF traces), it was discovered that the insert program did so many inserts that by the time the read only user executed the logic to analyze the error and CLOSE and reOPEN, the insert program had forced a second CA split that caused a second secondary allocation!

The user's first attempt at correcting this was to have the insert user recognize a CA split and issue an STIMER for 500ms., reasoning that 500ms. should be long enough for any read only user after these records to execute the CLOSE/OPEN logic. While this worked for a time, the user then implemented subtasking in the application (forgetting about the 500ms. timing factor). It was then discovered, or rather rediscovered, that 500ms. was too great a time when 50 subtasks were involved.
To 'fix' this problem, the user decided to switch implementing ALL processing in the same address space, doing away with SHR(2,x) completely. Unfortunately, this caused abends because the user forgot that DEB checking catches a CLOSE issued to another tasks ACBs, so the user would up recoding the application a third time. After that, everything worked. My point is that you just can't get away from the fact that you need to know about VSAM sharing BEFORE you code your application!

Getting back to the problem of read integrity with CA splitting, you must understand that you are trying to control an essentially serially reusable resource, and the time honored method of doing it is to use ENQ/DEQ. But the problem with ENQ/DEQ is that of itself, it does not inform you that secondary allocation has occurred. It will keep you from trying to retrieve a record while the insert is in progress, but what happens after the insert is completed? The user must devise a method of communication that is easy and that requires few, if any, restrictions, especially in MVS with its multiple address spaces. One method is to use ENQ/DEQ like VSAM does, i.e., choose a name that means something to all users. Then an ENQ (.....),RET=TEST could be issued to determine the status of the cluster. You would issue the ENQ only if you failed to retrieve the required record, i.e., in your error recovery procedures. While technically feasible, this idea has some major shortcomings:

- What will read only users do when the insert user completes processing and terminates, removing all ENQs?
- How will the read only users stop the insert user from doing an insert during the retrieval?
- If OPEN/CLOSE logic is to be implemented in the error recovery procedures of both types of users, how many times will it be executed, i.e., what is the quality of the recognition procedure to determine if an insert was done, is being done, etc.?
- If you are the insert user, how do you know that a split is in progress, or that secondary allocation will be required to satisfy an insert?

Let us look at the last problem first because it will be of general interest. There is a control block called the DIWA (Data Insert Work Area) chained to the AMB. At offset x'04', bits 0 and 1 respectively, there are flags that indicate that a CA split is in progress and/or that a CI split has been performed. There is also another flag in the PLH at offset x'03', bit 5, but it is for CI split indication only and a separate bit is not available in the PLH for CA split. The problem with both the DIWA and the PLH is that technically, these blocks are released after the CI/CA split has been completed and are considered meaningless by VSAM. (You can tell if the DIWA is 'in use' by examining offset x'01' - if in use it will contain a x'FF' put there by a TS instruction. The only way to conclusively prove that a PLH is in use is to follow the buffer chains and examine the flag settings in the BUFC.) At the time of writing, the DIWA flag indicating CA split was not erased when control was returned to the user - it was erased on the next entry to the split modules. Obviously, if you use this information in an application, you may have to rewrite the application at a later date if VSAM decides to clear flags BEFORE returning to the user.
An alternative is the journal exit - JRNAD. This exit sets up register 1 to point to a parameter list, the 21st. byte of which contains a reason code. Reason code x'20' is a CA split, and is available to all users, even though the VSAM Programmers' Guide indicates it appears only to users of shared resources. Now VSAM could change the code to agree with the manual, but since the JRNAD is taken during a split, you can use JRNAD to examine the DIWA yourself! This would appear to be a 'safe' method of obtaining the fact that a CA split has occurred.

Recalling that I have answered only one objection to the ENQ/DEQ logic, let me now continue with that method. The only way to answer the other objections is to make some assumptions about the way the insert user will issue ENQ/DEQ and proceed accordingly. Suppose, as an illustration only, the insert user does the following:

1. ENQ Assume a CA split will always occur.
2. PUT This is the insert.
3. test This is a test in JRNAD for a CA split. If a CA split occurs, a user flag is set.
4. DEQ This DEQ is executed only if the flag set by JRNAD is off.

The read only user has agreed to code the following:

1. GET If successful, skip everything below.
2. ENDREQ Invalidate data buffer to prevent VSAM lookaside. Remember that in non-create mode, the sequence set buffer is not invalidated.
3. VERIFY This handles the CA split with no secondary allocation problem. Recall that CNV processing is required for this macro.
4. POINT This positions to a different CA so as to invalidate the sequence set buffer. (Not needed if the RPL is same one used for VERIFY.)
5. flag Set a recursion flag to zero. This will prevent later loops.
6. GET This is the retry. If a no record found is indicated again, test the recursion flag. If not zero, you came through here already and it's a valid no record found. If the flag is off, turn it on and proceed.
7. ENQ If the ENQ indicates there has been no CA split, turn off the recursion flag and indicate a valid no record found. Otherwise, continue.
8. STIMER Allow for the insert to be completed.
9. refresh CLOSE and reOPEN your cluster.
10. Go to sequence #6.

Criticism.

The use of STIMER was described earlier, and while this should work in simple cases, the multi-tasking implications were discussed previously. To net it out, STIMER is obviously time dependent.
Then there is the problem of the CNV mode required for the VERIFY. Unfortunately, you cannot OPEN an ACB for CNV processing together with some other type, and since verification of a different ACB is meaningless, you either have to be willing to use CNV processing for the whole application or define a second ACB that performs subtask sharing with the first ACB. (Either use the same DD card for both ACBs or specify DSN sharing in both ACBs.) Otherwise, to issue VERIFY, you would have to CLOSE and reOPEN your cluster.

One can debate the merits of CLOSE and reOPEN endlessly. They are very time consuming, and if they must be used frequently, the application will be severely degraded. This would happen if the application EXPECTS no record found as the normal return. (One such application might be searching files for police records.) I must admit however that this is the exception. If you decide that the frequency of CLOSE and reOPEN is acceptable (theoretically, it is necessary only on secondary allocation), you might elect to simply forget about VERIFY and use CLOSE and reOPEN for all no record found errors. In the above sequence, you could then eliminate nos. 2,3,4,6,7,and 8. Sequence 10 would be a GET and a test for the recursion flag.

The last criticism is idea of always issuing ENQ/DEQ if you are the insert user, even though you know that 99% of the inserts will not cause any splits at all. Again, this is a question of timing. Unfortunately, it doesn't matter if it's 99.999%; if the possibility exists, it must be provided for in the application.

Rather than use the STIMER approach, a more positive idea is the use of a control record or control file. As an example, suppose the first CI is set aside as a control record and 'preformatted' with a single record that fills the entire CI. The preformatting would remove any possibility that the CI ould be split, and if you position this CI at RBA 0, even a CA split will not move the CI. This means you can always retrieve this record using either a key of all binary zeros or an RBA of zero. Assuming that your control record contains information about the number of extents in your cluster, a generalized sequence for the single insert user would be:

1. ENQ — This is an ENQ on the control record for exclusive control.
2. PUT — This is the insert.
3. test — If feedback code x'16' is returned with Reg. 15 = 0, a CA split has occurred. If feedback code x'04' is returned with Reg. 15 = 0, secondary allocation was required. If either of these feedback codes occur, update the control record.
4. DEQ — This is a DEQ of the control record.

As far as the read only user is concerned, the most probably course of action would be to first try to retrieve a record, and if successful all error recovery is naturally bypassed. If unsuccessful, the error recovery routine can do the following:

1. ENQ — This is an exclusive ENQ on the control record.
2. GET — Retrieve the control record and test for new extents. If new extents are indicated, go to sequence #7. (The ENQ is left outstanding to prevent the insert user from executing.)
3. VERIFY — Assume error involves adding to end of key range.
4. GET — This is a retry of the original request. If successful, the feedback code is zero. If not successful the feedback code is not zero. In either case, continue below.
5. DEQ — Allow the insert user to process.
6. exit — This exit presumes the feedback code has either been modified by the error recovery processing, or the feedback code has been set to zero by a successful retrieve.
7. refresh — This is the CLOSE, followed by the OPEN and another GET.
8. Go to sequence #5.

Notice that the use of the control record eliminates any time dependencies. I have omitted the POINT processing here on the assumption that the VERIFY is using the same ACB, i.e., CNV processing. If that is not the case, you will need to position yourself after step #3. This logic was explained in the previous example.

WARNING:
This logic assumes you do not intend to share the DASD between multiple CPUs that are loosely coupled. If you need to use shared DASD, you will have to issue your own device RESERVE/DEQ logic. This is covered later in this handout.

Final Comment on SHAREOPTIONS(2,x).

Regardless of what method you employ, there is bound to be some degradation in performance. The ENQ facility is implemented by SVCs, and so is OPEN and CLOSE, so the addition of these macros to a processing loop will obviously slow down your response time. You might make an assumption that each of these SVCs require 50,000 instructions to execute (whether they do or not), and proceed from there. It is better to inform users that their response time will be 7 seconds and come in at 4 then the other way around!

SHAREOPTIONS(3,x)

IF YOU HAVE NOT READ ALL OF THE PREVIOUS PAGES OF THIS DOCUMENT, GO BACK AND READ THEM NOW. SEVERAL OF THE PRECEEDING PRINCIPLES WILL BE APPLIED HERE!

The VSAM statement about this share option is quite simple and direct: VSAM offers no integrity assistance with share option 3. This means that VSAM will allow multiple users to OPEN a VSAM cluster and will allow those users to execute any properly coded macro against the cluster in any sequence. If the user should care not to implement some sort of control, VSAM will allow two CA splits to be in progress simultaneously. It would then be possible to actually destroy records by CIs being overwritten by actions in some other P, R, or AS. In plain English, the added problem here is that there is no write integrity and of course, no read integrity either.
BY DEFINITION OF SHARE OPTION 3, IT IS THE USERS RESPONSIBILITY TO PROVIDE INTEGRITY!

In dealing with this option, let me set up two restrictions which I will later attempt to remove.

1. There is no possibility of secondary allocation because the cluster was defined without it.
2. You loaded the cluster with FSPC(0,0) and filled all remaining CAs with 'dummy' records in an attempt to prevent CA splits. (CI splits would also be prevented as a whole CI could become available due to ERASEing all of its records, but the sequence set record is not updated to indicate a free CI except in the case of shortening or deletion of a spanned record.)

Now it is really not possible to prevent CA splits by technique #2 above because the user can still ALTER additional volumes into the definition of the cluster. Because VSAM allocates the primary space quantity on every volume (yes, that's a change from DADSM), you could get more space allocated to your cluster in this manner. While I admit it is insane to permit such antics in dealing with critical clusters, my point is that there is no VSAM flag, option, or coding that will absolutely prohibit this kind of thing. IT IS THE USERS RESPONSIBILITY TO IMPLEMENT EXTERNAL CONTROLS AS WELL AS INTERNAL CONTROLS! If your external controls are a little weak, then you had better examine your backup and recovery techniques - do they really work? Have you tested them?

Case 1. CA split is not possible due to restriction 2 above plus tight external controls.

First, I want to deal with the notion that putting a single 'high key' in the cluster will 'prevent' inserts that cause CI and/or CA splits, a technique commonly employed in ISAM processing. Naturally this will not work in VSAM because while a key of all x'FF's is allowed, that in no way prohibits either CI or CA splits.

Read that second restriction again. Even if you use FSPC(0,0), load a lot of dummy records, and then ERASE the dummies later, remember that VSAM uses a compressed key in the index and that ERASE does not cause recompression of the index. In general, every interval will still have its generic high key in the sequence set after ERASE (the one exception deals with spanned records). It has been said to me several times that this is a shortcoming of VSAM; I look at it as foresight, because if you consider that many new laws relating to right of privacy have been passed which, in turn, means that data processing cannot presume to use such things as social security number as an index, the idea of key compression has materially reduced DASD space requirements already!

This may lead you to believe that you now have generated the perfect cluster for sharing purposes - no CI or CA splits, no high level index extensions, no high used RBA changes, etc. Now, if you do all this, is your cluster really USABLE? What about the time you are going to spend reorganizing it when need additional space?

Now I don't want you to feel it's invalid to limit VSAM features. What I want you to think about is the simple fact that limiting certain VSAM features can make sharing easier, but making sharing easier has no effect if the application no longer does what it was intended to do. Further, the question of performance is a serious one. If you must have absolutely the fastest possible response time in every situation, then some limitations will have to be made, but it has been my experience that many applications require the 'ultimate' in response time only because the user community perceives that 'computers should be fast', not that the fast response time is actually required. If you are one of those users who started with only one display tube which had instant response, which 'degraded' to 5 second response time when you added the other 59 tubes, perhaps a little education is in order for those users of the tubes. I'll bet that if you can guarantee consistent response time of say 2 seconds, most of your users will accept it.

Now the first 'case' here is actually historical. ENQ/DEQ was used extensively in ISAM and BDAM (BDAM offered to do the END/DEQ for the user in the BDAM exclusive control option) to handle the situation of update but no insert. ISAM users quickly found out that they could not ENQ on the record (as they had been doing in BDAM) because there were multiple records in one block, so they used a system of double reads:

READ	Read the block containing the ISAM record.
ENQ	Usually on the first record so as to hold the block.
READ	This always reread the block because ISAM had no lookaside at all.
update	
WRITE	No deferred writes were possible in ISAM READ/WRITE logic - it was all BISAM.
DEQ	

I should point out that ISAM users tried to use QISAM for updates with in core indicies, but ran into the same problems as the present VSAM user. The idea that ISAM was better that VSAM in this respect s some sort of myth.

If you have any ideas of using the two restrictions previously discussed and ENQ/DEQ to control simple updates only, then you must clearly understand the following points:

1. You must ENQ/DEQ on a minimum of a CI, not a record. This means you must either know (or be able to compute) the CI number, or enter into a system of double retrievals as described above. You must do this because ENQs to different RBAs can still target the same CI.
2. If you plan to use the double read idea, you must deal with VSAM's lookaside feature. Lookaside says that the VSAM system will first determine if the record in already in memory in either a buffer associated with your string or in a buffer that is not associated with any string. Since you have just read the record, it is highly probable that the record is in memory. Thus, you will have to use ENDREQ to force VSAM to refresh your data buffer. Then you can reread your record and expect to get the latest copy.

One of the easiest ways to get a CI number, and thus eliminate the double reads, is to record the CI number as you load the cluster in some other data set. This idea is very old (when used with ISAM it was known as DISAM - Direct ISAM), and forms the basis of most data base products. When compared with the amount of time spent doing the double reads, it really isn't so bad at all, especially when you consider that if the number of CIs in your cluster is reasonable, the 'other' data set will be quite small. You may be able to keep sizable portions of it in memory at all times.

Now there are some maintenance consequences. If your cluster is a KSDS, and if your controls are weak, someone could ALTER in more volumes and permit CA and CI splits. Once this happens, you would have update the second data set with new CI numbers for the records moved. You can guard against this possibility in a program which inadvertently performs an insert by installing journaling; the journal exit is NOT entered for every record that VSAM moves during a split however - it's entered only for the record you are processing. Any other RBA changes must be computed by the user!

I know my readers are going to question an 'inadvertant' insert - how could that possibly happen when you are involved in something as heavy as sharing? Well, it did. The scenerio goes something like this. A large account assigned a task force to convert a major application, all in COBOL, from ISAM to VSAM. Many programs were involved and testing proceeded in the normal manner - convert one program, test it, convert a second, test it, integrate one and two and test it, etc. Now to 'get around' what the account thought was 'heavy' VSAM education, only two people involved in the whole account (account numbered in the 100 programmer range) received any training in VSAM at all. One of these people wrote COBOL source for native mode VSAM that was to be copied by all the other programmers involved in the conversion.
Several months later, after many serious discussions with IBM representatives, the account claimed that VSAM was losing data and because of that, and a few other problems like being over a year behind on maintenance, the conversion to MVS was slowed. The day I got there, I was informed that they had just discovered an 'inadvertant' insert buried in an error recovery routine in a 'read-only' program. Why the chief programmer had not discovered this in initial testing is speculation.
In this particular case, journaling might have helped to detect the problem earlier, although I have to admit that journaling might constitute a rather drastic step for this kind of problem.
My point here is: EVERYONE CONCERNED WITH AN APPLICATION INVOLVED WITH SHARING OF ANY KIND, VSAM OR OTHERWISE, NEEDS TO BE INFORMED OF DECISIONS THAT YOU MAKE THAT AFFECT SHARING. You just can't have uninformed people making assumptions about how your company is using sharing.

To get back on the track, the reread method of old ISAM will still work if you agree to use ENDREQ so as to stop VSAM's lookaside. Since you want VSAM to physically reread the CI for you, BUFND should be obviously set to the minimum - 2. (VSAM always reserves one data buffer for insert activity, so setting BUFND=2, which is the default, will not cause two CIs to be read every time.) Since you are not doing inserts, you don't need to worry about index buffers at all, but I should reiterate that two reads are twice as expensive as one - your overhead will double. While you will not need to do any journaling, you might still want to at least consider the two data set approach even though you are only updating. You might even go as far as using a KSDS to hold your CI numbers and an ESDS to hold the actual data - after all, you are only updating. (You can't use an ESDS if your update includes changing the length of a record and/or deletion. But those activities are really forms of insert and one of our restrictions is no insert. Did you understand that no insert also meant no record length change and no deletion?) The sequence of macros would be:

1.	GET NUP	If you are read only, you are finished. If not, continue below.
2.	*SHOWCB	Get the CI (or CA) number.
3.	ENDREQ	Force VSAM to invalidate your data buffers.
4.	ENQ	On the CI or CA previously obtained in step 2.
5.	GET UPD	If you switched to a second string, be sure you issue the ENDREQ to this string. If not, you must issue a MODCB* to switch from NUP to UPD.
6.	PUT UPD	Put the record back.
7.	DEQ	

* Frequent use of macros like SHOWCB and MODCB can be time consuming. You may want to consider the use of the CBMMs (Control Block Manipulative Macros) described in the Options for Advanced Applications. These are available only in Assembler and will require more effort on your part, but the reduction in overhead is significant.

There are at least two ways that you can greatly simplify all of this if your sights can be lowered a bit. First, you could divide your data set into key ranges and enqueue on a key range instead of going to all the trouble of getting the CI or CA. VSAM provides up to 255 key ranges so that means you could ENQ/DEQ on as small as 1/255 of your cluster without using actual CIs or CAs yourself.
If you decide to use key ranges, you should be aware of two things:

- A key range is defined by additional fields in your clusters' catalog records. This can significantly increase the space requirement in a catalog.
- If you define a cluster with the KRNG attribute, the primary space quantity is allocated on all named volumes at DEFINE time, not on an as needed basis.

(At least two accounts I know didn't realize that last point. Programmers coded CYL(50,50) and KRNG and operations didn't know what happened to all the space on their 3350s. I don't want to relate the whole story, but you can be sure people were mumbling: "It's that &*ç%S# VSAM again!"

Second, you could simply enqueue on the whole cluster. If your update rate is low, this can be a very reasonable approach, but remember, if you do this, and your update rate increases significantly, you will probably have to do some recoding later. ANY recoding of anything to do with sharing is a SIGNIFICANT change. Have you so informed your manager?

Case 2. CI split can occur, but not a CA split. (You prohibit secondary allocation, but allow FSPC. However, at load time, you force at least one record in every CA through careful calculation of space requirements.)

The only real change from Case 1 is that you must enqueue on a CA instead of a CI. This is due to the fact that if you didn't, User A operating in one P, R, or AS could ENQ on CIx in CAx and User B, operating in a different P, R, or AS could ENQ on CIy, also in CAx, and the ENQ would be O.K. in both instances. If both users then did a CI split, VSAM would 'target' the same free CI in the CA for both users. Whichever user wrote back last would overwrite the other users data. Since the sequence set is also written back during a split, the integrity of both the index and data components is now compromised.
If you decide to ENQ on the CA, IT IS YOUR RESPONSIBILITY TO ENFORCE YOUR CONSTRAINTS! Remember, you can add volumes to a VSAM cluster through the ALTER command without rewriting the cluster.

Case 3. CA splits are allowed, but still no secondary allocation.

The problem here is more complex because any CA split can 'target' the first available CA, so enqueue on the CA is not effective. Thus two users could overwrite each other if both were involved in a simultaneous CA split. This is more disasterous than Case 2, if more disasterous has any meaning.

You might be able to use the key range idea discussed in Case 1 at the top of this page, but you will have to use the ORDERED attribute to prevent VSAM from searching for space on any volume but the one assigned to that key range. Further, you will need non-overlapping volumes. However, this will allow you to at least divide your cluster into parts. The only other way is to lock out the whole cluster.
Another problem to consider is the use of index buffers. If BUFNI is sufficiently large, VSAM will bring the entire index set into memory and leave it there. Now it is possible that a CA split requested in a P, R, or AS other than your own could have necessitated an addition to the index set. If both users have all of the index set in memory, the change in the index set will not be noted by the user not requesting the split. Admittedly, this is unlikely in a controlled environment, but as of this writing, there is no macro short of CLOSE and reOPEN that will force VSAM to refresh

index set buffers. Since the VERIFY macro uses the normal index buffer algorithms, you cannot expect it to refresh index buffers for you either. About the only legitimate method you can employ to force VSAM to refresh index buffers is to simply set BUFNI=1. Then, whenever a direct request is issued for retrieval, VSAM is forced (because all direct requests process index levels from the top down, even if that will result in the same sequence set record as now occupies the buffer being reread) to read the highest level of index into the buffer, then the intermediate, etc. You should definitely use REPLICATE if you decide to set BUFNI=1; that will minimize DASD search time. It goes without saying that good data set placement on the DASD volumes would greatly affect the performance of the application as well.

Since there is no way short of CLOSE and reOPEN to force VSAM to refresh index buffers if BUFNI is greater than 1 and you are using SHR(3,x), the only way you could permit simultaneous inserts from multiple P, R, or AS is to use some type of ENQ/DEQ yourself and then signal all other users to CLOSE and reOPEN.

Now between those two positions (BUFNI=1 or CLOSE/OPEN) there are probably a hundred ways to handle concurrent update/insert IF YOU ENFORCE SOME LOCAL RESTRICTION. One example would be to never add more records to your cluster than would fit in the existing levels of index, i.e., the highest level index CI is never split. You could then think of using BUFNI=2 (STRNO=1) so as to keep the highest level of index in memory. Your performance would be better, but you would still need to signal everyone when you did a CA split so they could CLOSE and reOPEN, but the signaling would have to be done only if there was a change in the highest level of index.

As you can tell, I am splitting hairs here. If you state you wish to do concurrent or simultaneous inserts from multiple P, R, or AS, then let me reiterate what you must consider:

1. You must start with empty data buffers to prevent VSAM lookaside in the data component. Normally, this is achieved with an ENDREQ.
2. You must start with the most recent version of the index. Since there is no VSAM macro that will force index buffers to be invalidated, you must:
   - A. Use BUFNI=1 so as to force VSAM to reread all index levels. This works only in direct mode.
   - B. Use your own system of signaling such that you know if anyone else has done a CA split. If they have, you CLOSE and reOPEN.
3. VERIFY your cluster so that you will pick up high used RBA changes if only CI splits occurred in the last CA. (If you CLOSE and reOPEN, you obviously don't need this.)
4. Issue your ENQ. This probably means that you are going to look at some VSAM control blocks so as to find out where the new record is targeted. If you decide to use regular VSAM macros like SHOWCB, then you will have to first condition VSAM by say issuing a POINT, extracting the information, ENDREQ to free the VSAM RPL, ENQ on your selected field, then the insert, etc.
5. After the insert is over, you must see to it that the buffers are actually written. This is automatic in direct mode, but not in sequential. If you decide to use advanced options like GSR or LSR, VSAM permits deferred writing of even direct inserts. Use of these options (GSR or LSR) is beyond the scope of this section.

At this point, you should realize that the key element here is not really VSAM at all - it's that signaling between P, R, or AS. Now you are going to need some information to signal:

- A. The number of active participants.
- B. Whether the participants received your signal.

The best way to handle such signaling is either with user written SVC routines or the control record described in SHAREOPTIONS(2,x). If you use ENQ/DEQ, you must be able to answer the question of timing, i.e., how do users know when a particular operation is in progress, when it ends, and what it did?

Assuming you decide to use BUFNI=1, a possible macro sequence is:

ENDREQ*	Invalidate all data buffers from the last I/O operation.
VERIFY	Make sure the high used RBA is correct.
POINT**	Obtain information for next macro.
ENQ	Indicate to others what you are about to do.
ENDREQ	Force refresh of data buffers.
GET/PUT	This is the update/insert. If a GET, record is reread due to ENDREQ so you get the latest copy. Because BUFNI=1, the index buffers are also refreshed.
DEQ	

* This macro is just a 'safty valve' and is not necessary unless you have done an update GET without the corresponding PUT and you use multiple ACBs with subtask sharing to do the VERIFY.

** You can obtain the RBA of the record by issuing a SHOWCB against the RPL. The high allocated RBA and the RBA of the last used byte (not the same as the RBA of the last record) can be obtained via SHOWCB against the ACB. You can also use the DIWA as explained on page 10. Recall that VSAM may have invalidated your buffers after a request (GET NUP). Use of the DIWA may involve recoding later if VSAM internal change.

If you decide to use BUFNI greater than 1, you will have to do something about forcing a refresh of the index buffers. (One method of doing this was indicated on page 11.) Before you do any of this, you might want to think about using IMS or CICS instead. While I have no doubt that many of my readers can code an application that runs faster than either IMS or CICS, I also know that many times the effort involved is measured in man YEARS! Please - be honest with yourself - do you have that kind of manpower? Do you believe IMS was written 'overnight'? I have called on some people who wanted to write just an interface to data base products, and found that a 'simple' job mushroomed into years of work, not to mention maintenance.

Case 4. Secondary Allocation allowed.

This case was covered in SHAREOPTIONS(2,x) on page 9 and following. The discussion there resulted in the requirement for some kind of control record and the requirement of using CLOSE and reOPEN due to the fact that many internal control blocks like the ARDB will have to be modified. Depending on the complexity of your code, the response time can degrade severly - in one case it was 1000%, although that was an exception. But you begin thinking in terms of 400 - 500%.
I point this out here because SHAREOPTION(4,x) described next also will degrade throughput, by perhaps the same amount, but perhaps that will be acceptable if you realize that SHAREOPTION(3,x) is not a lot faster if you have to handle all possible situations.

You can save a lot of trouble by restricting the number of VSAM options allowed, using combinations of KSDS and ESDS, and perhaps even using RRDS if the application lends itself to fixed length records. You must be able to enforce your restrictions however, and all code must be checked to insure that no one inadvertantly used an option incorrectly. Finally, remember maintenance. Have you informed them about your restrictions?

SHAREOPTIONS(4,x)

Think back to the more difficult points we covered in SHR(3,x). What were they?

CA splits.

SHR(4,x) prohibits all CA splits.

VSAM lookaside in direct processing.

SHR(4,x) always invalidates buffers in direct processing. Hence, VSAM refreshes the data buffers for all direct processing.

CI splits in the last CA, i.e., a high-used RBA change.

SHR(4,x) prohibits a CI split if the high-used RBA would have to be changed.

While this may look like the perfect answer to all your problems, you should consider that:

1. SHR(4,x) will not do any ENQ/DEQ for you.
2. The forced buffer invalidation will degrade the performance significantly over other share options. This must be taken into account if you plan to do a lot of direct processing.

The whole controversy in the field over share option 4 came about because of the different implementations in DOS/VS and OS/VS. The DOS/VS system uses a 'track hold' feature of the DOS/VS system to handle SHR(4,x). The DOS/VS system also uses the GETVIZ area which is similiar to our PLPA. The track hold feature is not available in OS/VS systems and the GETVIZ idea is available only in MVS (at the time of this writing) as an option called CBIC - Control Blocks In Common. Unfortunately, the overhead in the OS/VS systems to handle the required internal manipulations is higher. This has led to people saying that the DOS/VS share option 4 is 'faster' than the OS/VS version. This is true; where it turns out to be very serious is in a DOS/VS to OS/VS conversion when the DOS/VS system made extensive use of SHR(4,x).

I might point out that the DOS/VS manuals state that their internal techniques do not provide read integrity - only write integrity. Switching to SHR(3,x) in OS/VS and doing some recoding would provide both read and write integrity, but of course any recoding in a conversion simply adds to the already big conversion, so it's tough to swallow. There is no easy answer to this situation.

You should understand the following points as well with respect to share option 4.

1. Since the data buffers will be automatically refreshed on all direct requests, there is little value in using anything other than the minimum for BUFND. (The minimum is 2.)
2. If you attempt to do a CA split or a CI split in the last CA (high-used RBA change), you will get a return code of x'1C' as a feedback code.
3. Even if you decide to use BUFNI greater than one so as to keep portions of the index set in memory, there is no need to 'worry' about index splits because share option 4 prohibits high-used RBA changes.
4. You can pick up sme performance in the direct processing area if you elect to use CNV processing. (To do that, you must code CNV in the MACRF field of the ACB and also in the OPTCD field of the RPL.) If you use CNV processing, you obviously agree to process the contents of a CI yourself, so you obviously need some way to remember what is in each CI. Whatever way you choose, it's totally up to you.
5. You still need to do the ENQ/DEQ logic and if you allow inserts that do not change the high-used RBA, you will need to ENQ on the CA, not the CI. As described earlier, locating the 'target' CA for an insert might be difficult. Again, that's totally up to you.

Before leaving share option 4, reflect on the fact that your major problem will be determining the proper CA or CI to enqueue upon. Since this option prevents CA splits and CI splits that change the high -used RBA, you can record the CA number of a record in a second cluster. Later, you can use this information to control your ENQ/DEQ logic. If the idea of two data sets does not appeal to you, you can use an old BDAM trick to eliminate the second data set:

1. Load the cluster allowing 'n' dummy records to written in an agreed upon position in the cluster - usually at the beginning in BDAM. In VSAM, you could put the dummies at the end and use the read backward feature of VSAM to retrieve them later.
2. During the load, the CA/CI information for each record is maintained in a table in memory. At the conclusion of the load, the cluster is updated by writing the table into the dummies.
3. Retrieval consists of always retrieving the 'dummies' first and then using the CA/CI information therein to perform the enqueue.

Your only 'liability' with this approach - if you can call it a liability at all, is you must code the programs to load and maintain the table.

Final Comments on Cross Region Sharing.

You should realize by now that the complexity of cross region sharing will decrease exponentially as you agree to limit the facilities that particular cluster will provide to the ultimate user. Further, if you can predict the content of CAs, (by carefully organizing them with respect to CA and CI size, FSPC, IMBED, etc.), the difficulties that sharing anything, not just VSAM, will decrease. Make an honest appraisal of the following:

- Does the user really need all the facilities of VSAM or are you just trying to implement the facility 'because it's there'?
- Would addition DASD storage alleviate the problems you run across, perhaps by allowing the shared clusters to reside on their own unique volume or volumes? If so, why not make the case for more DASD space known clearly to your management. Mostly reasonable people will listen to carefully prepared arguments, especially when you include statements about the complexity and maintainability of the proposed sharing.
- Document everything well. This is the place for too much documentation instead of too little.
- Don't make the mistake of assuming that VSAM is a data base - it is not; VSAM is an access method. The type of recovery offered in an IBM product like IMS/VS is not offered in any access method, VSAM included. You will be wasting everyone's time if you write letters to IBM demanding backout and recovery to be put into VSAM so they work 'like IMS'. Instead, set up your own backout and recovery procedures and test them thoroughly.

Cross System Sharing.

There are only two options for cross system sharing - SHR(x,3) and SHR(x,4). As with SHR(3,x), SHR(x,3) does nothing except allow the user to access a cluster now in use by another system. Access to a cluster requires access to the cluster's catalog by the rules of volume ownership, so the first questions to be considered here revolve around catalog sharing.

VSAM allows the UCATS (user catalogs) to be shared by multiple systems. You accomplish this by doing an IMPORT CONNECT of the catalog name into the master catalog (MCAT) on the systems you want to participate in the sharing. Even though the definition of a catalog does not allow the SHR keyword, VSAM internally will allow catalogs to be shared between systems by utilizing the standard shared DASD support. (This is the RESERVE support that users of shared DASD may be more familiar with.) Note that VSAM does not offer any more than currently permitted by the shared DASD option and there are certain restrictions. The restictions invole the MCAT. Simply stated:

- A single MCAT cannot be the MCAT of two or more operating systems.

This does not mean that a MCAT cannot be a user catalog to another system; it can. Thus the MCAT of system 1 can be a UCAT to system 2 while the MCAT of system 2 can be a UCAT on system 1. In the past, many users shared critical system libraries among multiple systems and so I am always asked why VSAM has this restriction on the MCAT; the individuals asking the question always insist that: "If I want to destroy my systems, why don't you let me?" These people should remember:

- IBM is committed to data integrity. We are not going to knowingly allow a simple command like IMPORT CONNECT to have the power of destroying systems.
- In virtual systems, the critical page data sets are cataloged in the MCATS. In MVS, these page (and also, for that system, the swap) data sets are themselves VSAM clusters with all the volume ownership rules applied. Allowing a system to share another system's page data sets would be an integrity exposure of gigantic proportions.

Cross System Share Option 3.

First of all, there cannot be a mixture of SHR(x,3) on one system and SHR(x,4) on another due to the principle of volume ownership. Now if you use SHR(x,3), VSAM does not issue any RESERVE macros for you or any other macros. SHR(x,3) means you are on your own; a user claiming to be read only could go right ahead and do a CI or CA split with no warning to anyone.

You should realize that it is possible to define a PATH over a base cluster which forces read only access. While this may be satisfactory for many, the PATH concept is not an absolute guarantee because other users need not use the path - they can just as readily use the base cluster directly. If you have a LCMP - loosely coupled multi-processor - you also have a set of instructions (read direct/write direct) which you can use to communicate with the other processors; JES3 uses these instructions extensively. However, even if you use JES3, VSAM does not use these special instructions; you can use them yourself, but VSAM does not use them, period!

Thus, you the user are totally responsible for integrity in the SHR(x,3) situation. I have previously described the complications of sharing on a single CPU. Not the least of these problems was finding out what happened elsewhere. In a LCMP environment, you are totally responsible for all communications between CPUs with respect to VSAM SHR(,3). For this reason, I recommend that you limit yourself to read only users when implementing SHR(x,3).

Cross System Share Option 4.

As with cross region share option 4, this option prohibits CA splits and CI splits that would change the high-used RBA. Buffers are also refreshed for each request in direct processing. Now you are responsible in this share option for issuing the RESERVE and RELEASE macros to control the actual DASD sharing; VSAM DOES NOT ISSUE THE RESERVE AND RELEASE MACROS FOR YOU! (VSAM will handle the DASD sharing for catalogs, but not for user data sets.)

The net of this is that it works exactly like SHR(4,x) except that you are using RESERVE and RELEASE instead of ENQ and DEQ. I might add that RESERVE and RELEASE lock out the entire volume, i.e., they work on a physical DASD unit basis, not a cluster basis. In practice, this means you need to carefully control the content of the whole volume, not just the cluster, or else serious degradation can occur.

I see no need to reiterate all of SHR(4,x) here; the rules would be the same for SHR(x,4) as well. As of the time of this writing, there is no difference between SHR(3,4) and SHR(4,4) for the user.

As a final note here, you should be aware that the RESERVE macro puts flags in the UCB; it does not modify the channel program. When the channel program is executed by IOS, the flags in the UCB are recognized and IOS does the actual reserve. Obviously, if the channel program is not executed the reserve is never set. At least one user tested his program by using //X DD DUMMY statements. Since no actual I/O is done for a DUMMY data set, the RESERVE/RELEASE logic was never tested. When the program used real I/O, ... well, you know what happened.

ESDS, RRDS, and the Share Options.

I deliberately left ESDS and RRDS clusters out of the sharing discussions so far because I wanted you to 'get into' the problems associated with sharing a KSDS cluster. Now neither an ESDS nor an RRDS cluster has an index component, so if you could use these types of clusters, a major part of your coding could be simplified.

Earlier, I mentioned using two clusters handle the problem of keeping track of the location of a record in a CI. Well, if the cluster was an ESDS cluster, you could easily keep track of a record's RBA by putting the record's key in a very small KSDS along with it's associated RBA. Since the KSDS would be small, the chance of splits would be much reduced, and even if you totally lost the KSDS, you could rebuild it quickly by simply reading the ESDS. Data base products like IMS/VS use these techniques extensively. If the 'key' in the KSDS is compact and numeric, you could substitute RRDS for the KSDS which would be even better since there is no index component in an RRDS.

This technique would offer you practically unlimited expansion through VSAM's powerful secondary allocation facilities and you could also ALTER in more volumes as you need them without reorganization. Compare that to ISAM! You would also retain the capability of using records of varying size. You would lose the deletion facility of KSDS however; no deletion is possible in ESDS except a logical deletion that you could implement through a flag of your own design. This means that if you wanted te reclaim space occupied by deleted records, you would have to reorganize the KSDS/ESDS or RRDS/ESDS combination. I suppose you can look upon that need for reorganization as some sort of negative, but if you are negative to begin with, this whole paper will be a negative to you so I need not say any more. I don't think it's a negative. In fact, if you organize you backup cycle on reasonable terms, you reorganization can be fit in nicely into that backup cycle to produce a neat package.

The RRDS cluster offers some interesting facilities. While you cannot use an RBA to retrieve a record in an RRDS cluster, you can use the generic form of the relative record number because the relative record number is treated as a key, not an RBA. You can also delete a record in an RRDS cluster. On the other hand, the RRDS cluster is limited to true fixed length records. Note that if you used an RRDS/ESDS combination, you could use the generic key capabilities of the RRDS cluster to aid you in locating the RBA of the record in the ESDS cluster. Since the purpose of the RRDS in this case is to 'index' the ESDS, all the entries in the RRDS would be fixed length - a perfect case for RRDS.

Now both the KSDS and RRDS could need to be extended in these situations, so the rules discussed previously about ENQ/DEQ would still apply as would all the share options. But the KSDS and RRDS clusters should be quite small, simplifying the ENQ/DEQ logic. (You could ENQ on the whole cluster if it's small and the number of extensions correspondingly small.) But beyond that, a major advantage is that even if you lost your 'index' totally, you can rebuild it by simply reading the ESDS! Think about that!

Multi-tasking and Sharing.

Before getting into this, let me state that I am asuming that you are going to use just one string in each ACB for the time being. This means you coded STRNO=1 in the ACB and your code doesn't cause a dynamic string addition to be done. The actual sharing is controlled by the parameters DDN and DSN in the ACB. Three cases are possible.

1. You use a single DD statement for all ACBs. The coding of DDN or DSN in the ACBs doesn't matter if you do this - they are equivalent.
   VSAM will provide full integrity in this case.
2. You use multiple DD statements, but the DSN parameter on the DD statements reflects the same data set name. Further, you coded DSN sharing in the ACB.
   VSAM provides full integrity in this case also.
3. You use multiple DD statements, but code DDN sharing in the ACB.
   VSAM uses the options coded in the SHR parameter to control this situation, i.e., this is identical to sharing cluster between P, R, or AS.
   Since this was treated previously, and all the same rules apply, it will not be treated now.

Cases 1 and 2 are what most people mean when they use the term 'multi-task sharing'. The basic implementation of multi-task sharing is to chain your AMB to the AMBs of existing users through the AMBL. (AMB is Access Method Block and AMBL is Access Method Block List.) This is key because in doing the chaining, the buffer pool of the original user is utilized for all of the chained AMBs. Since the integrity flags are maintained in the BUFC, and since there is a single set of BUFCs, VSAM can offer full integrity within the P, R, or AS.

Let us assume you are using either case 1 or case 2. Now even though you coded STRNO=1 in each of your ACBs, the net effect of your sharing of the buffers is as if you had one ACB with multiple strings. If you keep that in mind, the rest of this will be easier. (VSAM builds all necessary control blocks, including required BUFCs, when you OPEN the ACBs.)

One of the first things that is difficult for a beginner is the concept of dynamic string addition vs. exclusive control of a string. When a beginner first hears about dynamic string addition, the feeling is that 'I can't code anything wrong - if I do, VSAM's dynamic string addition will fix it for me'. This question is more easily answered if you first consider what exclusive control means in VSAM sharing within a P, R, or AS. Suppose you have two strings and one of them issues a direct GET for update. The sequence set is scanned, the CI read into an available data buffer and the BUFC associated with that data buffer is marked 'in exclusive control'.
When the second string also requests a record for update in that CI, the VSAM lookaside routines will find the buffer (actually the BUFC) marked 'in exclusive control'. Since VSAM understands that to mean that an update is going to occur for that CI, and since reading another copy of that CI into memory could cause an integrity violation as described in the previous discussion of share options, VSAM will return a feedback code to the second requestor (x'14) informing that requestor that the record (actually the CI) is in the process of being updated. Presumably, the application is coded to handle such a situation.
Note that dynamic string addition cannot be used to solve this 'problem' because rereading the same CI into some other buffer doesn't change the fact that the CI is in the process of being updated. Dynamic string addition is useful only when an entirely new request is presented to VSAM - a request that could be serviced except for the lack of buffers and associated control blocks.

The next hurdle is: "Why not let VSAM do all dynamic string addition? I'll code to handle the exclusive control, but let VSAM handle the STRNO automatically." When the STRNO is grater than 1, VSAM allocates the control blocks and buffers in contiguous storage. If dynamic string addition is performed, the extra control blocks and buffers are not necessarily contiguous with the first set. Since some of these control blocks, and of course the buffers themselves, must be page fixed for I/O, the performance of your application could be degraded due to excessive page fixing if dynamic string addition was used. The net of this is that there is nothing like knowing what you are doing, calculating the proper requirements, and identifying them to VSAM. This is a little off the subject I suppose because I stated earlier that each ACB had only one string in my example, but the technique is perfectly general so I thought I'd review it here because I always get a lot a questions about this subject.

Are there any liabilities in multi-tasking or multiple string VSAM sharing?

Yes. When you OPEN a VSAM cluster, OPEN builds a DEB for each component of your cluster on your task's DEB chain. Now VSAM doesn't make much use of these DEBs, but the OS/VS system expects them to be on the proper TCB chain when you either CLOSE the cluster or abnormally terminate. A CLOSE issued against a cluster from the wrong task will ABEND because CLOSE processing checks to see if the DEB is properly chained to your task. The problem comes up like this:

- An old application is converted from ISAM to VSAM.
- The old application allowed everyone to define their own DCBs.
- The old application had a number of service routines provided by the main task so that users didn't have to recode a lot of duplicate instructions. The service routines invariably include I/O routines.
- To OPEN and CLOSE the ISAM data sets, the users may or may not invoke the service routines. (The usual case is that they invoke the service routine for OPEN so as to inform the main task that they exist, but are allowed to CLOSE by issuing their own CLOSE.)

In all the cases of this I have delt with, all the users admitted that their code was poor and represented 'quick and dirty' fixes to past problems, yet VSAM was totally 'blamed' for their recoding difficulties. As a matter of fact, this is not a VSAM problem at all - it's not even a problem. It's an operating system that has closed some of the loopholes.

The implication here is that if you do multi-task sharing, you must CLOSE the ACB from the same task that OPENed it. At least one user implemented a system of his own control blocks, but failed to intercept abnormal subtask terminations. His service routines would eventually try to CLOSE the ACB for the now non-existent task, causing an ABEND of the main task. Another user realized that some type of inter-task communication was necessary, but he didn't want to code it, so he put all the ACBs in the main task and passed their addresses to the subtasks. The main task OPENed and CLOSEd all right, but he found that subtasks would abend leaving buffers in exclusive control. (Since there was no DEB on the subtasks DEB chain, no CLOSE was issued in abnormal termination.) Further, since he didn't know how many tasks were up, dynamic string addition was adding as many as 50 extra buffers at a time. Experiments with STIMER to limit a subtask's access to the VSAM ACBs was fruitless and the application is now run with the understanding that it will 'bomb' from time to time. As usual, all of this trouble is blamed on VSAM.

A second concern is the area of abnormal termination. During abnormal termination, a CLOSE function is executed by the operating system against DEBs on the subtask's DEB chain. Notice I said CLOSE function, not CLOSE macro. The functions performed by this automatic CLOSE at abnormal termination are not the same as those performed by the CLOSE macro issued by the user, and they never have been! If you don't believe this last statement, (and a lot of people I meet don't), try updating a sequential data set that is blocked with the PUTX macro and APEND before you CLOSE. See if your last blocks

that were updated but not yet written back are physically updated i.e., see if abnormal termination writes them back. The point is that abnormal termination is not going to write back the contents of buffers because abnormal termination doesn't know that buffers are buffers; the buffers themselves may have been the cause of the abnormal termination and the system has no way of knowing which buffers are valid and which ones are not. Obviously, if a subtask terminates abnormally and it is using VSAM, requests marked 'in exclusive control' by the terminating subtask might not be released. This may eventually cause the main task to abnormally terminate when dynamic string addition tries to add strings to compensate and there are no bytes left in virtual storage to do the job.

This issue can be addressed by the main task ATTACHing the subtasks with either the ECB or ETXR parameters on the ATTACH macro. These parameters permit the maintask to be informed of the subtask's abnormal termination. When so informed, the maintask can CLOSE the necessary data sets to clean up the control blocks. To do this, the maintask must be controlling all the I/O.

A third concern is the area of access to the ACB and related control blocks, i.e., what if DO NOT do subtask sharing. If you elect not to use subtask sharing, then separate sets of control blocks are built in your P, R, or AS. These will react to share situations just like SHR(3,3). The difference is that the users don't have to code interregion communication into the program; all parts of all tasks in a specific P, R, or AS are available to all of the tasks within that P, R, or AS. A specific example is the ENQ/DEQ facility where users are allowed to use the SMC (system must complete) option at the step level. (SMC is not allowed in MVS at the system level.)

Recommendations for Subtask Sharing.

If you plan to use VSAM in a significant multitasking application, you should control all the VSAM (and other I/O too) I/O from the job step task and use the ECB and/or ETXR parameters on the ATTACH to become informed as to how your subtasks terminated. Further, ESTAE and/or ESTAI exits in MVS (STAE/STAI in VS1 and SVS) should be employed to control error detection and handling. At the outset, these measures appear overly complicated, but later ease in application coding will more than justify their expense.

In the past, new users of OS/VS operating systems have reacted quite violently to these suggestions. Some common complaints were:

"I was never told all of this."
"We didn't do any of this in our old system."
"It worked before."
"You need a lot of education to use VSAM"

The significant parts of all of these statements is the fact that most of these people assumed that:

- You could apply old techniques to VSAM.
- Sharing of anything can be accomplished by coding a parameter somewhere.
- Recoding an application so as to use sharing is a simple task.

I want to point out just one case of how this kind of thinking can lead to some really complex situations. This user started out using what he called 'total' subtask sharing. He never checked if his definition of subtask sharing was the same as VSAM's definition. It wasn't; his 'total' was VSAM's no subtask sharing, but this person coded a major application using this premise. Since he employed a very large machine to test the application, it ran for some time without 'error', although it was noticed at the time that much more virtual storage than anticipated was being used, and that the storage was being used by VSAM. This was chalked up as 'VSAM buffer inefficiecies' and the application went into production.

In production, even more virtual storage was used. It was then found that as many as 50 tasks were 'sharing' a VSAM data set; in the planning stages it was thought that the maximum number of tasks would be 'around 10'. The user now decided to put all the I/O handling (but not OPEN and CLOSE) into the main task because he had done that in some other application and it 'saved a lot'. Now since almost every retrieval was a retrieval with no update, and there were never any inserts of any kind, all of these measures seemed to work for a time. At this point, the coder had no formal VSAM training of any kind - he had 'got it out of the book'.
Then the application started abending about once per week. Investigation found the following:

- All the subtasks were supposed to do OPEN and CLOSE; all OPENed but not everyone CLOSEd.
- None of the subtasks did any error handling, nor did they use any form of ESTAE or STAE.
- Some of the subtasks were updating. Some of these updates involved record length changes but the subtask coders had not been told that a record length change is equivalent to an insert.

The next step was to remove all OPEN and CLOSE logic from the subtasks, put a single ACB in the main task, and pass it's address to the subtasks which would use it to modify RPLs in there programs. This was viewed as a significant improvement because the virtual storage was significantly reduced. As time progressed, it was noticed that whenever subtasks abnormally terminated, it was likely that the main task would abnormally terminate later that day with a lot of dynamic string addition indicated. (An abnormally terminated subtask was simply reATTACHed 'until it worked'.)

When management finally got into the act (many months too late) and the facts about VSAM sharing became known, the application was still not recoded; management felt that 'something could be done' to salvage 'all that good code' without resorting to 'advanced techniques' like ETXR routines or ESTAE logic. As of this writing, the final decision was to simply let the application abend and restart it when necessary.

My point then is you can't blame VSAM for an individual's ignorance. In fairness to the individual I described above, he inherited an application that had been designed years ago for a model 40 running BOS - do you even remember BOS? It used it's own roll your own T-P access method and was really a giant service module for an essentially uncontrolled group of users represented by the subtasks.

Let me summarize some of the important ideas associated with subtask sharing.

1. Subtask sharing must be implemented by coding DSN in the related ACBs or using the same DD statement for all the ACBs. If you don't do one of these two, you DO NOT use subtask sharing.
2. If you don't use subtask sharing, then multiple ACBs in a P. R. or AS react just like SHR(3,3). If you permit multiple uncontrolled CA splits in that kind of environments, you can and will destroy your cluster.
3. The key resource shared is the BUFC. (Flags related to I/O activity are in the BUFC.)
4. If a requestor gets a feedback code of 'record held in exclusive control', dynamic string addition will not be employed by VSAM to 'solve' this case. It is up to the user to handle this situation.
5. VSAM maintains a DEB for each OPENed VSAM component on the proper TCB DEB chain. All OPEN and CLOSE macros must be issued from within the proper TCB.
6. Consider the use of CS and CDS instructions to achieve your own type of ENQ/DEQ processing.
7. MVS does not permit SMC in system mode; it is allowed in step mode.

8. Any type of sharing is difficult code. It's no disgrace to admit you don't know how to do it and seek help.
9. You should be aware that later recoding of any application that uses sharing of any type will probably be quite difficult; shun any 'quick' approaches to data set sharing.
10. Be sure management is cognizant of the essentially details of any shared data set application.
11. If you elect to do less than a complete job of handling sharing, be sure to document what you did and why you did it.

I think most of this is plain common sense. Don't you?

Other Methods of VSAM Sharing.

Whether you are programming in VS1, SVS, or MVS, you should evaluate the address space design philosphy of MVS before proceeding with a major VSAM sharing undertaking. In MVS, fetch protection is standard and inter address space communication must use SRB scheduling which, in turn, requires authorization. Further, MVS supports TCMP - Tightly Coupled Multi-Processing. This means that two tasks can be in simultaneous execution. This means that any dependence on not using fetch protection, concurrent task execution only, etc., could lead to a complex recoding job if you move to MVS later.

As stated earlier, IBM offers extensive VSAM sharing through the use of program products like IMS/VS. These products commonly use special SVC routines to aid the sharing effort. Before you decide to do the same thing, i.e., code your own share routines, you should seriously consider one of these products. Isn't it more productive to use a product that shares the VSAM clusters correctly and concentrate on organizing your own data to advantage rather than spending months developing your own sharing scheme? If you answered 'No!', read on.

First, it must be understood that a user SVC can issue a GETMAIN for any area, even SQA. With that in mind, the first problem to be delt with is how to recognize when the sharing routine you coded, hereafter termed the 'driver', is available for use. Presumably, you put the driver in the nucleus, the PLPA, or in it's own address space. Further, you started the routine running, much like starting an IMS Control Region. The starting of the routine can be handled by an operator start command. Remembering that an SVC routine can run in supervisor state, key 0, and authorized, the driver can acquire storage in a common area and set up control information that is understood to all users. One way to accomplish this is to use the word in the CVT reserved for the user (CVTUSER) to point to a user CVT - call it UCVT - where all required information is stored.

If a user SVC - call it SVC zzz - is used by all users to communicate with the driver, it can simply abend the requestor if the driver is not up when the requestor makes the initial request. If the driver is up, the initial request should identify the user to the driver so that the driver can terminate the users in the event the driver itself fails. Our SVC zzz would now operate as follows:

1. On initial request, 'join' the network of users by filling in and/or creating additional control blocks chained to the UCVT.
2. On subsequent request, the SVC zzz would send the I/O request to the driver. It is likely that the SVC zzz would take care of the necessary WAIT/POST logic commonly employed in these situations, noting that in MVS, the WAIT/POST logic would have to employ cross memory POST.
3. Some method of indicating the final request would have to be devised so that the user's control blocks could be freed. Unless this was done, the driver would not be able to be stopped.

Of major concern in this method is the location of the ECBs that will be necessary to establish the actual communication between the driver and the user. These are necessary because the SVC zzz is really a setup function; it does not set tasks dispatchable - POST performs that function. It is likely that a pair of ECBs will be necessary - one to inform the driver that work exists (and upon which the driver WAITs), and a second to inform the user that the work is done (and upon which the user WAITs). Remember that MVS also checks that the location of ECBs is addressable by the address space and that the ECBs have the correct content when a WAIT is executed against them.

If you want to implement this idea, you must decide exactly how much information will be set up in the user CVT and associated tables, where it will come from, and how you are going to get rid of it in the event of an abnormal termination. The last item, abnormal termination, is the one frequently neglected. Remember that your user CVT is going to indicate, by it's very presence, that a user has joined the 'network'. Failure to delete user information will mean that at the very least, the user will not be able to rejoin the 'network' later. It can also mean that the driver is not able to be stopped and could therefore lead to a re-IPL!
In the event you wish to be able to terminate the user due to an incorrect request or other error condition, you will need an ID for each user. Several VS1 users hit upon the idea of using the storage protection key of the partition as the ID since it was unique for all user partitions in VS1. Beware! Neither SVS nor MVS use unique storage protection keys for individual users. It is better to extract the normal systems ID (the one used by the operator to cancel a job) for this purpose.

IBM implements user SVC logic in HASP, JES2, JES3, TSO, TCAM, VTAM, and IMS/VS, to name a few. In VS1, the JES system also uses SVC logic, but JES in VS1 is integrated into the control program so it's hard to see 'user' code. To write this paper, I used the TSO system because the SVC there must execute in the user region or address space and communicates to a 'driver', much like you would have to do. TSO uses two SVCs to simplify the coding. You could get away with one. The TSO logic is quite plain if you read the microfiche.

If you coded all of this, you would have developed what is called a subsystem. Basically, a subsystem is a module or modules that establish and maintain a protocol between a user of the subsystem and the subsystem itself for the purpose of obtaining a specific function from a generalized operating system. Theoretically, if the generalized operating system were radically changed, you could still obtain the desired functions by recoding only the subsystem. It's a lot of work to do it - probably thousands of man hours. It might be worth it to you if you anticipate extensive use of the functions, but it might also be just as financially sound to use program products, tight scheduling procedures, or new or redesigned hardware configurations to achieve the same results. Only you can decide.

Common Errors.

Here is a list of common errors I have experienced. By a wide margin, the reason for the errors has been the feeling that since VSAM is an access method, no education is required to use it. Remember, you the reader are getting education right now, but what about the rest of the people at your account? Especially, what about junior programmers today who will become senior programmers tomorrow? Will you afford them the opportunity to get the same or better education as yourself, or will your account tell you to 'pass it on'? Some people just can't teach others and some of us just don't have the time to teach others, so are you depending too heavily on that 'pass it on' idea?

1. Assuming that abnormal termination will 'take care of everything', including writing back buffers that were scheduled for update.
2. Trying to pass the addresses of ACBs to other tasks for I/O. OPEN and CLOSE require the same task to own the DEB and abnormal termination of a subtask operating this way can leave a buffer in exclusive control forcing unwanted dynamic string addition later.
3. Failing to understand that an update that changes the record length is equivalent to a delete followed by an insert of the record. These kinds of updates can cause CI and CA splits.
4. Assuming that the catalog records for your cluster are maintained while you are processing. Catalog records are maintained by the CLOSE macro.
5. Assuming that since the catalog must be updated by dynamic space acquisition, your AMDSB in the catalog records for your cluster will also be updated at that time. Updating the AMDSB is the responsibility of CLOSE or CLOSE TYPE=T.
6. Assuming that all index CIs, including sequence set records, can be forced into main storage and kept there permanently. Only index set records can be kept in storage.
7. Failing to provide adaquate abnormal termination facilities, especially in multi-tasking environments. This is really an enlargement of point # 1 above. This means use of the ETXR exit in the main task to capture subtask failures and the use of ESTAE or STAE for the main task.
8. Failure to understand that an operator cancel is considered an abnormal termination.
9. Failure to provide proper backup and recovery in the event of abnormal termination. This is especially true in light of point # 8 above. If an operator cancels a VSAM job, a recovery procedure should be defined to validate the clusters that were in use by the job. Repeated restarts without recovery can lead to loss of data.
10. Failure to understand that read integrity can be lost even though write integrity exists.
11. Failure to understand that loss of read integrity can lead to records becoming non-processable by the program doing the reading if another program has moved the records during a CA split. In other words, read integrity is not a performance issue - it's an integrity issue.
12. Assuming you can modify VSAM control blocks directly i.e., without the use of MODCB or it's equivalent CBMM. (CBMM - Control Block Manipulative Macros - these are described in Options for Advanced Applications.)
    These macros do not just go directly to the field in question and operate on it; they start at the block you specify and perform extensive checking to make sure, for example, that if you change the record length, flags are set to accomodate the change.
13. Assuming there is no difference between the first release of VSAM and enhanced VSAM.
14. Assuming you can do everything that VSAM allows using the ISAM interface.
15. Assuming that compiliers support 100% of the VSAM capability or will in the future. While IBM commonly enhances it's compiliers from time to time, there is never any guarantee that the enhancement you are waiting for will ever occur.

The ISAM Interface.

It has recently come to my attention that some customers are assuming that as long as a VSAM feature can be coded in the DEFINE the ISAM interface will make use of it. Specifically, a customer felt that since SHR was a parameter of DEFINE, it could be used to control the ISAM interface in the same way that native VSAM would use the parameter.

Basically, you should view the ISAM interface as providing most of the facilities of ISAM through a VSAM cluster, but no facility over and above what ISAM provided are added. As an example, to 'extend' an ISAM data set, you had to code DISP=MOD. You don't have to do this for VSAM, but the ISAM interface enforces the rule. As for sharing, ISAM used the DISP parameter on the JCL statements or, in the event multiple users wanted to process an ISAM data set concurrently, all users coded DISP=SHR in the JCL and totally controlled the sharing themselves, i.e., there was no SHR option like VSAM's in ISAM.

Now the VSAM SHR option operates through feedback codes to the user at OPEN time. Note that there is no implied WAIT; if you code SHR(1,x) or SHR(2,x) and there are already users opened for output, VSAM will return a feedback code to you at OPEN time (x'A8') and you will have to determine what to do with it. VSAM neither waits till the proper conditions exist nor abends.

Since ISAM had no equivalent facility, the ISAM interface cannot map the VSAM return and feedback codes for this situation into an equivalent facility. Therefore, the DCB is not opened. VSAM will normally inform you of the problem through messages (of the type IECxxxxI) but VSAM will definitely not wait until conditions are favorable and attempt to reopen your DCB! Users of ISAM were supposed to check DCBOFLGS to determine if OPEN was successful or provide the equivalent test if COBOL or PL/I was being used. Failure to do this checking could lead to unpredictable results. If you checked this flag in ISAM and issued a user abend for a DCB not opened, you will do the same in the ISAM interface mode of operation.

If you plan to share VSAM clusters through the ISAM interface, I would recommend you do the following:

1. See if the use of DISP=OLD, i.e., initiator data set name enqueue lockout, will suffice.
2. If not, you will have to use DISP=SHR and go to SHR(3,x) or SHR(4,x) in the DEFINE. Since the functions of the ISAM interface are not directly controllable by you, some external controls on the cluster must be used to limit your liability i.e., prohibiting the use of secondary allocation, tight scheduling of jobs that plan to share the cluster, etc.

If neither 1 or 2 above appeal to you, I suggest to use native mode VSAM instead of the ISAM interface. Naturally, this means some recoding, but if you consider the amount of time you would spend trying to modify the ISAM interface to fit your needs, and the subsequent time you will spend maintaining those modifications, I think you will agree that native mode VSAM will turn out to be cheaper and faster in the long run.

Special Sharing Options in MVS.

Two different special VSAM cluster share options are provided by MVS which will allow the VSAM user to control the cluster from multiple address spaces, BUT, with severe restrictions. The options are called:

Global Shared Resources (GSR), and
Control Blocks in Common (CBIC)

Obviously, the objective is to get a single set of VSAM control blocks in a common area of MVS (almost always CSA) so that VSAM record management routines can be used to handle data integrity. Note that while the use of these options can produce full integrity across address spaces, they do not improve cross system sharing problems except insofar as the numbers of control block chains involved in the cross system sharing are decreased.

Global Shared Resources.

GSR is the older of the two options and is used heavily in IMS/VS. To use it, the following restrictions must all be met:

- You must operate in supervisor state when you reference the GSR control block structure.
- You must obtain storage for the control block structure in Subpool 231 (CSA).
- The control block structure must be in one of the priviliged keys (0 through 7).
- You must build (via GENCB) all required ACBs, RPLs, and EXLSTs in Subpool 231 and also format your buffer poor (via BLDVRP) in the same subpool.
- The exit routines associated with the control blocks should be commonly addressable i.e., in PLPA, MLPA, FLPA, or CSA.
- GSR is mutually exclusive with CBIC and ICI (improved control interval processing).

The VSAM publication OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications - GC26-3819 list additional restrictions with regard to coding various macros.

The intention of the designers of GSR was that a single address space would handle all the actual I/O to the cluster; other address spaces would perform I/O to the cluster by forwarding that request to the address space that built the control structure - the so called 'control region'. However, if all required routines were located in common storage and the location of the ACB and it's related control blocks was known to a second address space, that address space could perform it's own I/O. (Recent enhancements to IMS/VS allow just that sort of thing to happen.)

The reason for people wanting to do their own I/O in this situation is that any communication with another address space in MVS requires SRB scheduling. Doing your own I/O would eliminate the SRB scheduling overhead. Unfortunately, the first restriction is difficult to implement in multiple address spaces - supervisor state. It is relatively easy to get into supervisor state (MODESET macro) but do you want to allow ALL users to run in supervisor state - even part of the time? One possible solution to this dilemma is a user SVC which was covered previously in this paper. Another solution might be CBIC.

Control Blocks in Common.

CBIC starts out looking very similiar to GSR. The restrictions are:

- The address space that OPENs the CBIC structure must be in supervisor state.
- The address space that OPENs the CBIC structure must be in a system key (0 through 7).
- All exit routines must be commonly addressable.
- All address spaces that reference the CBIC structure must be in the same key as the user who OPENed the structure.
- The ICI option MUST be used.
- CBIC is mutually exclusive with LSR and GSR, a VSAM checkpoint cannot be taken, and CBIC cannot be used to process the catalog, page or swap data sets, system data sets, CRAs, or the VVIC (Virtual Volume Inventory Control) of the 3850 Mass Storage device.
- Only one address space may OPEN and CLOSE the CBIC structure.

Notice what is NOT a restriction - the other users do not have to be in supervisor state! Further, IBM builds the proper structure in CSA for you - no need for GETMAIN to CSA and the requirement for cleaning up CSA if the address space that OPENed the structure fails. (CSA is NOT automatically cleaned up by IBM after an address space abends. The user is supposed to accomplish that using either ESTAE or better, one of the user exits in RTM.)

Other users must be in the proper key however. You might use multi-tasking to do that:

1. Begin processing by getting in supervisor state. (MODESET.)
2. ATTACH a subtask in the proper key that does all CBIC handling.
3. Turn off your authorization and get back into problem state.
4. ATTACH the user as a subtask in the normal key. Data can be passed from one task to a task keyed differently by changing the key of the subpool. The CHNGKEY macro can accomplish that.

There is no CBIC option in the ACB. To get VSAM to use CBIC, you must modify the ACB yourself by setting bit 2 of the field ACBINFL2 (x'33') to a '1'. The other little trick is that you must use ICI, Improved Control Interval processing, also know as 'fast path scheduling'. ICI eliminates most of the VSAM checking and ALL OF THE STATISTICS and does not allow journaling. If you need any of these, you will have to code them yourself.

By using the technique of getting started in supervisor state and then turning off your authorization and relinquishing supervisor state, you can use the CBIC option to effectively allow multiple address spaces to fully share a VSAM cluster.

Both the GSR and CBIC option require a fair amount of prior thought and coding on the part of the user. IMS/VS and CICS/VS use many of the techniques described and my first advice to you would be to make use of these products if at all possible.

Final Hints, Comments, and Miscellaneous Information.

One.
Be sure all people who come into contact with a VSAM sharing situation understand basic VSAM terminology. In a recent experience, operations had the wrong idea of the meaning of the word 'load'. They allowed two jobs that 'just updated a few records' to run concurrently with disasterous results. I have heard the phrase "We are only going to educate the key people....." so many times it makes me sick. ANYONE even close to a VSAM sharing application had better know what the terms mean.

Two.
Be sure to be at the proper maintenance level for your system. If you are a year behind in maintenance, you will be spending all your time trying to figure out which PTFs to apply, only to find some other problem occurs after the fix is applied. I myself am unable to comprehend how a large multiple CPU user can fall that far behind in maintenance, but I have experienced it and the experience is 100% bad!

Three.
Be sure to allow the IBM PSR sufficient facilities for obtaining the necessary dumps. Realize that a problem in an online application can mean that a stand-alone dump may be necessary before problem resolution can occur. If that online application has a 100% up-time requirement, what are you going to do about getting dumps? Normally, it is impossible to debug without a dump. It's no fun if you are the customer on the horns of the online dump dilemma.

Four.
Be sure you get your facts straight. I spent a month investigating a VSAM problem where three customer and two IBM representatives swore that VSAM was producing an 037 ABEND. Finally, it was 'discovered' that VSAM was really not producing a dump at all, that it was a message with an 037 reason code, and that the module producing the message was an OPEN module, not VSAM. The dump resulted from a program issued ABEND macro! A lot of time could have been saved here by just stating the problem correctly.

Five.
Consider your plan for backup. I consider your plan no plan at all unless you have:
- at least two qualified people trained in VSAM at the system programmer level.
- coded and tested every backup procedure on live data, including 'standalone' steps.
- educated all users of VSAM in the importance of keeping the standards you have set up.
- educated chief programmers on the value of implementing recovery procedures WITHIN an application instead of just backing up packs every day.
- educated operators on the destructive force of a CANCEL command, especially the new MVS FORCE option of cancel.

Six.
At least one customer I know has modified VSAM code to 'streamline' it's checking. This is a very serious undertaking. I advise you to sit down with your IBM representatives and discuss precisely how the modified code will be supported. The problems here stem from the fact that while many mods to MVT are well known and are relatively easy to dispense with in problem determination, changes to MVS espesially are not at all easy to understand and can easily exponentially increase problem determination time.

Catalog Sharing.

Unlike other forms of sharing, catalog sharing is totally automatic, whether you want it or not. This has performance implications in multi-CPU environments. If you examine the options that are 'defaulted' when a catalog is defined, you will see that SHR is SHR(3,3), but no matter, IBM still performs integrity checks.

The reason catalogs can be shared is that VSAM builds some special control blocks in common areas of the system. Since the control blocks are in common, VSAM can moniter all catalog requests. Let's examine both the control blocks in common and the ones in your P, R, or AS to see how they relate.

Common Blocks

AMCBS — Access Method Control Block Structure

The AMCBS contains the location of the MCAT at x'04' (the CCHH), The location of the control block manipulative macros at x'0C', the location of the first CAXWA at x'14', and for MVS, information about VSAM shared resources.

The AMCBS is built at NIP time and for catalog sharing, the key field is the location of the CAXWA.

CAXWA — Catalog Auxiliary Work Area

This is an area that contains many fields, but basically, the CAXWA contains information about the status of a catalog that is currently open. A very important field is at offset x'0B' which is a use count field; later you will see how this field controls the number of requests that can be active to a catalog at one time.

At offset x'20', a set of fields exists which correspond to the information in type L catalog control record. Again, you will see later that this information causes the TSO user to have to LOGOFF before a new entry in the catalog will be recognized in certain circumstances and other interesting oddities.

Blocks built in the users' P, R, or AS

PCCB — Private Catalog Control Block

The PCCB represents a catalog defined by the user via a JOBCAT/STEPCAT or, in MVS, by dynamic allocation.

At offset x'1C' is the catalog's DSNAME and at offset x'48' is the alias used to locate the catalog if any.

It is important to note that a PCCB exists for CVOLs as well as UCATs in MVS.

The PCCB directs the order in which VSAM will search catalogs in MVS; in VS1 is represents the order of concatenation used in JOBCAT/STEPCAT.

Note that there is no PCCB for the MCAT, even if you include the MCAT's name in a JOBCAT/STEPCAT. The MCAT is always located via the AMCBS and so the order of searching MCAT cannot be controlled via JOBCAT/STEPCAT.

Both the CAXWA and the PCCB are chained to other CAXWAs and PCCBs respectively, but there is no direct vector from one chain to the other. The basic VSAM search algorithm is to search your PCCB chain to see if you have already allocated the required catalog. If you have, VSAM will use it, and will know where it is because it can locate the correct CAXWA via the catalog's DSNAME, which appears in the CAXWA at offset x'34'. In MVS, if the required catalog is not found in the PCCB chain, VSAM will use dynamic allocation to add to your PCCBs. There are exceptions to this simplified explaination, but they will come later.

Opening VSAM Catalogs.

Since a catalog is conceptually a KSDS cluster, an OPEN must be executed to build an ACB and related control blocks. When you open a normal KSDS cluster, VSAM uses the information in the cluster's catalog records (type C, D, and I) to establish the necessary control blocks. The importance of the AMDSB was previously described in this respect.
But a catalog can be allocated to multiple users at the same time by having multiple PCCBs - one for each user that has the catalog open. When these users CLOSE the catalog, CLOSE processing would normally update statistics in the catalog, but in the case of catalogs, if other users are still open, updating the catalog's statistics is meaningless. This means that the statistics in the catalog's cluster record (a part of the catalog's SDR - self describing records) are meaningless.
To handle this situation, the special catalog control record was devised; it is always CI #3 in the catalog's SDR and is obtained and updated via EXCP processing. Now the critical point here is that the CCR is NOT updated when a single user closes a catalog - it is updated when VSAM detects that ALL users of a catalog have finished with that catalog. There are exceptions to this 'rule', one of which is EOV processing when a catalog is extended, but in general, you should now understand that:

1. When a catalog is opened, the use count in the CAXWA is incremented.
2. Statistics about the catalog are maintained in the CAXWA.
3. The statistics in the CAXWA may not update the CCR when you close the catalog.

The statistics I am talking about here are statistics about what is in the catalog, not how big the catalog is at the present moment. The exact description of the statistics is: (offsets are for the CAXWA)

x'20' CI number of the highest allocated CI in the catalog.
x'23' CI number of the next free catalog CI.
x'26' Number of deleted CIs.
x'29' CI number of the first deleted CI in the catalog.

Obviously, if the field at x'29' has not been updated properly in the CCR, later catalog commands like LISTCAT may not give all the entries now in the catalog. It all depends, of course, on whether the processing uses the information in the CAXWA (accurate), forces VSAM to update the CCR before executing (accurate), or simply reads the CCR and uses it (inaccurate).

In examining the CCR still further, you will discover that it contains the high-used and high-allocated RBAs for each component of the catalog. You will also note that this information is NOT in the CAXWA. This means again that unless VSAM updates the CCR, the information is inaccurate. VSAM issues the VERIFY macro just before updating the CCR; by doing so, the high-used RBAs are updated in the control blocks associated with the catalog's ACB. Before going further, you should understand that this method of handling the CCR accounts for certain phenomenon that may have happened in your account to date:

1. An entire catalog is REPROed and in the same step a LISTCAT is executed. The LISTCAT shows only the catalog's SDR.
   Explaination:
   The catalog's CCR will not be updated until the last user closes the catalog; since you are still in the same step, the catalog is still allocated and open.

2. A TSO user allocates a new catalog and places entries in it, but the catalog cannot be used by others until the TSO user logs off the system.
   Explaination:
   While the new catalog is obviously there, it is located via searching in the master catalog. Until the MCAT's CCR is updated, the new entries in the MCAT cannot be located by other system components.

Closing A VSAM Catalog.

Much of this information has already been described in the open processing description. In closing a UCAT, VSAM simply decrements the use count in the CAXWA; if that use count is zero, VSAM understands that you are the last user of that catalog and will update that catalog's CCR. Recall that VSAM always issues a VERIFY macro just prior to the update of the CCR to insure that the RBA information is accurate.

The master catalog is never really closed in the sense described above. When the system is shut down, the MCAT's CCR is updated just like any other catalog's CCR, but the point here is that there is nothing a user can do via JCL or VSAM macros and/or commands that will close the master catalog.

Sharing Implications of Catalog OPEN and CLOSE.

One.
Since the CAXWA is in storage common to all users, the catalogs are ALWAYS shared among all users on a single CPU regardless of SHAREOPTIONS.

Two.
As long as a catalog remains open, the critical catalog control information is in the CAXWA and related control blocks. This implies that a user who OPENs an ACB for catalog processing (either codes the catalog's DSNAME on a DD statement or codes CATALOG=YES in the ACB) must issue the VERIFY macro to update the RBA and related information. Failure to do this causes unpredictable results.
Note: If you intend to OPEN an ACB for catalog processing, the Options for Advanced Applications (GC26-3819) lists other requirements under the heading "Opening a Catalog".

Processing a VSAM Catalog.

The exact method of processing a VSAM catalog request depends to some extent on the number of buffers allocated to the catalog and the size of the index component of the catalog. The IBM default BUFFERSPACE used when you DEFINE either MCAT or UCAT is 3072. IBM further assumes your index component has only two levels - sequence set and highest level - for the 3072 bytes are allocated as follows:

1. Two strings are assumed. One sequence set buffer of 512 allocated for each string.
   Total is now 1024.
2. Each string needs one data buffer of 512 bytes.
   Total is now 2048.
3. One insert buffer is always allocated by VSAM for the data component.
   Total is now 2560.
4. The remaining 512 bytes are allocated to the highest level index.
   Total is now 3072.

VSAM will permit a user to specify BUFFERSPACE for a catalog at DEFINE time. If you specify a value over 3072, VSAM will attempt to set up more than two strings, to a maximum of 7 strings. The exact formula is:

Number of Strings (RPLs) = MIN(7,BUFFERSPACE/1K - 1)

If 8192 is chosen, there will be 7 strings. The space accounting is:

7 strings	1K each, total of 7K.
insert buffer	.5K, total of 7.5K.
HL index	.5K, total of 8K.

Notice that this means that even though you have a 'giant' catalog, the intermediate level index records are not resident in storage. This can cause serious loss of performance if the catalog is being used by multiple partitions, regions, or address spaces. Now even though the AMS manual currently states that BUFFERSPACE is limited to 8192, you can code a larger value. If you code 9216, there will be an 'extra' 1024 bytes after the 7 strings are set up which VSAM will use for index buffers. (You could code 10240 and get an extra 2048 bytes, which would allow an extra 4 index buffers, but it is rare for a catalog to need that many.)
Note that this is particularly important for MVS users with respect to the MCAT because MCAT is opened by NIP so you have no chance to code the AMP parameter on the JCL to override the buffer specifications.

WARNING:
YOU SHOULD NOT ATTEMPT TO DO THIS IF THE CATALOG RESIDES ON A SHARED DASD BECAUSE OF THE BUFFER REFRESH THAT IS ASSOCIATED WITH SHARED DASD, I.E., SHR(4,4).

(At the time of writing, specification of BUFFERSPACE larger than 8192 was not valid for VS1.)

If you are processing a UCAT via JOBCAT/STEPCAT, you can code the BUFNI in the AMP keyword on your JCL statement. VSAM will still allow a maximum of 7 strings, but the extra index buffers (if coded) will speed up processing if many requests to the catalog are required.

Note:
The LISTCAT command is a major user of the catalog.

VSAM uses keyed direct processing to access the HKR and CNV processing to access the LKR. This is partly due to the fact that the HKR is subject to CI/CA splits while the LKR is not. To write a new entry in the catalog, the LKR record is inserted first, using the CCR to locate a free CI. The HKR is updated next. The exact internals differ depending on whether you are inserting or retrieving.

LOCATE:

Issue an ENQ (shared) on a major name of SYSIGGV2 and a minor name of the catalog's DSN.
Attempt to acquire a string. If available, continue with request.
If no string is available, set the requestor waiting on an ECB to be posted when the current user has ended.

The purpose of the ENQ is to protect locate users from mixing with update users and retrieving invalid information. Why this is a consideration will become apparant when you read the UPDATE processing on the next page.

UPDATE Processing.

Issue an ENQ (exclusive) on a major name of SYSIGGV2 and a minor name of the catalog's DSNAME.

When the ENQ is honored, all other users are finished with their processing and the update user 'single-threads' the update through the catalog.

Summary of Catalog Sharing in a Single CPU Environment.

1. Any number of users can have a catalog allocated to them via the PCCB chain.

2. The key catalog management control block is the CAXWA and it is in common storage. All VSAM catalog processing funnels through the CAXWA.

3. VSAM supports a maximum of 7 strings for accessing a single catalog.

4. The strings allocated to a specific catalog will allow concurrent read access to the catalog, but update access is single-threaded using the regular ENQ mechandism.

5. All of the above occurs regardless of the coding of BUFFERSPACE and SHAREOPTIONS.

6. For performance reasons, the content of the CAXWA and related control blocks is not necessarily written back to the catalog when you close your catalog. (Actually, you neither OPEN nor CLOSE a catalog; the schedular performs these functions as part of allocation/termination.)
This may lead to a requirement for you to have another step in your jobstream (forcing the schedular to unallocate, then reallocate your catalog) if you are processing the catalog itself and you need the latest information. This would also imply that you do not allow multiple users to be using the catalog you are attempting to process, as VSAM does not update the catalog's CCR record until all users of the catalog are finished.

7. In a TSO environment, you may need to LOGOFF, then LOGON again if you are manipulating a catalog (normally, this means DEFINEing or recovering a catalog).

8. Since there is a maximum of 7 strings regardless of definition, there is a performance implication in having very large catalogs shared by more than 7 users at a time.

9. Since PCCBs are built for JOBCAT/STEPCAT statements and for dynamically allocated catalogs in your P, R, or AS, VSAM will search the PCCBs first in an attempt to satisfy a catalog operation not directed to a specific catalog via the AMS CATALOG parameter.
Further, the PCCB contains only two names - the name of the catalog and the name of the alias (if any) used to locate it. If the argument of the search has a first level qualifier of either the alias or the first level qualifier of the catalog's DSNAME, the search will stop at the PCCB.
If not, the system will either indicate an error (VS1 and SVS) or continue searching the MCAT (MVS) for other possible alias entries that match the argument's first level qualifier.

This strategy has performance implications whether the catalog is shared on a single CPU or is owned by a single user on a single CPU. If the preponderance of the requests all have the same first level qualifier (like a TSO userid), then performance can be expected to be good; if not, performance will suffer in a direct ratio to the number of user requests that need that catalog.

Sharing a VSAM Catalog in a Multi-CPU Environment.

As noted earlier, the CAXWA is a control block that contains very important information relative to catalog sharing. For this discussion, I assume you have the previous few pages that discussed sharing the catalog across P, R, or AS in a single CPU environment. The key field for this discussion is offset x'1C' in the CAXWA which contains the location of the UCB upon which the catalog resides.

If the UCB pointed to by the CAXWA is marked 'shared', VSAM ASSUMES YOU ARE SHARING THE CATALOG AMONG CPUS, whether you really are or not.

From the above statement, it is obvious that the coding of SHAREOPTIONS is meaningless with respect to catalogs. If VSAM notes that the catalog resides on a shared UCB, it:

1. Implements RESERVE/RELEASE logic for all catalog requests.
2. Refreshes all buffers with each request.

That's all there is to it! The implications, however, are quite serious.

In any operating system, RESERVE/RELEASE works on whole volumes only. If the volume upon which the catalog resides is a busy one, the catalog operations will simply add to the long queue times for that volume. The only really effective way to operate in this situation is to have the catalog and/or clusters on shared volumes, but not really share them with the other CPUs; in this case the shared DASD is more of a backup facility than a sharing facility. Experience teaches that the RESERVE/RELEASE logic adds that much overhead to the processing cycle.

In MVS, if an address space cannot get control of a resource (the RESERVE is actually a special case of ENQ), the AS can be swapped out. If that address space has issued a RESERVE but not yet issued the RELEASE (a special form of DEQ), it is possible for the address space to issue a second ENQ for another resource now unobtainable. Eventually, the AS could be swapped out, taking the 'volume' with it. This can ultimately lead to a 'chain' of address spaces swapped out, all because one address space could not access the catalog! This has been APARed, and may be fixed as you read this, but it does point up the problems of sharing catalogs among multiple CPUs.

It is also possible to IMPORT CONNECT the MCAT of one CPU as a UCAT on the other and vice versa in a multi-CPU environment. In VS1 and SVS, this adds the load of the RESERVE/RELEASE and buffer refresh, but that's all. In MVS, you can do the same thing, but MVS requires the page and swap data sets to be VSAM data sets catalogued in the MCAT. Imagine an MVS system waiting for a page or swap data set because the other CPU has issued a RESERVE on the volume!

At least one user has shared UCATs, but has modified the VSAM code to eliminate the RESERVE/RELEASE logic and the buffer refresh. All catalogs are nonrecoverable. Even with very good operational control (theoretically, in their shop, no two jobs EVER access these catalogs at the same time), they have lost the catalog from time to time. In several cases, the cause of the loss could not be determined. We can only guess that it is probably related to the fact that without buffer refresh and knowing how the CAXWA is used, one CPU probably had some info in it's copy of the CAXWA that simply didn't get written in time.

All these wonderfuls things can be yours, if you are willing to pay the performance price!

Sharing of CVOLs in an MVS Environment.

In MVS, the VSAM catalog structure supports CVOLS. They too have full integrity when shared between the user ASs, or among CPUs. The mechandism is to build control block structures in each address space and use ENQ/DEQ to serialize update requests. The update requests are single-threaded just like the update requests to a VSAM catalog, but CVOL processing allows an unlimited number of read requests. Possible extensions to the CVOL are recognized by using pseudo open/close logic each time an update is requested.

The CVOLS are shared among multiple CPUs in a manner exactly like VSAM catalogs, and they suffer the same problems.

As a point of possible performance improvement in this area, the extension check mentioned above must read the FMT1 DSCB in the VTOC since there are no self-defining records in a CVOL. By keeping the CVOL close to the VTOC, you can possibly eliminate some unnecessary seek time.

VSAM BACKUP AND RECOVERY

This section contains both an explaination of how certain key VSAM commands operate and ideas on how you might use them to effect backup and recovery of VSAM clusters and catalogs.

While the operation of a command can be verified by reading the SRLs, PLMs, and if necessary, the microfiche, the ideas are just that - ideas, or techniques.
I AM NOT RECOMMENDING THESE TECHNIQUES OVER OTHERS.
I AM NOT IMPLYING THESE TECHNIQUES ARE THE BEST, NOR EVEN BETTER THAN OTHERS.

My objective in this section is to point out to you, the student, all the problems that can befall you if you do not plan your backup and recovery carefully. Further, I will strongly maintain that an untested recovery procedure is no procedure at all. In short, no amount of ISP, book, or classroom training replaces the actual testing of your planned procedure. Therefore, reading this section is not enough. You must actually implement the ideas presented here before you can say you have a good recovery procedure.

Note.
In 1978 IBM announced that the utility IEHDASDR will be stabilized at some point in the future. Further, an SU was announced for MVS (84) called simply DEVICE SUPPORT. The initial publications on this SU indicate that the SU will perform all the functions formerly performed by IEHDASDR plus some others. The SU was a no charge SU at the time of this writing.
This SU does not effect the logic described herein; if you are using the new SU, simply read DEVICE SUPPORT wherever IEHDASDR is mentioned.
This new SU does not effect other IBM program products, IUPs, or FDPs that relate to IEHDASDR. They will still be supported until separate notification is given you by IBM.

General Comments and Background.

Whenever the subject of VSAM backup and recovery is mentioned, I here the question: "Why is it so complicated?" Well, it isn't really. Ten years ago backup and recovery wasn't even mentioned in most classes; the fact that a product worked at all was enough. It was understood that backup and recovery was the responsibility of the user. Furthermore, data processing was then in it's infancy. Like the calculator of today, if it broke, you waited until it was fixed and started over.

Today, we have non-professional DP users who are accustomed to teleprocessing, time sharing, remote job entry and other concepts that demand that a main processor be always available and that data be current. Further, the concept of a corporate data base is taking hold, a concept that for the first time means that the data is not in a file cabinate somewhere in case the computer goes down. Thus data processing has become highly visible - a small outage has immediate repercussions throughout the corporation.

Another aspect in backup and recovery is machine vs. software reliability. In the past, operations took backups of critical files. Since multiprogramming was not generally used until the late 60's, even the worst disaster meant that a single job only had to be rerun. In the late 60's and early 70's, multiprogramming came into widespread use, but most looked at it as simply running multiple machines. The idea of sharing data between concurrent applications was still not used by the average corporation. This meant that while emphasis was placed on hardware reliability, there wasn't much thought given to reliability in software, especially user written software. Like the vendor software of the past, the fact that it worked was enough. As these programs and design concepts are brought forward into the late 70's, the lack of adaquate backup and recovery starts to show, especially when data sharing is attempted concurrently. The net of these ideas is that VSAM gets blamed for difficult backup and recovery techniques when in fact, this might be the first time an account has formally considered recovery at all! In other words, it was going to happen anyway; VSAM just happened to be there when it did.

IBM's basic philosophy in the area of VSAM recovery is as follows:

- We are going to teach you about VSAM recovery in IBM classes. This in itself has sometimes led some to state it's harder. I guess it's harder when you compare it to nothing.
- VSAM will give indications when it suspects errors, but VSAM will not provide automatic recovery facilities to match these indicators.
- VSAM will provide utility commands to compare, locate, check, and with proper backup, recover errors.
- Nonvsam utilities will also check, where appropiate, for the presence of VSAM clusters (actually space) on volumes to insure that the VSAM rules are enforced. Thus, a utility like IEHDASDR will check VSAM passwords.
- VSAM IS NOT A DATA BASE! VSAM IS AN ACCESS METHOD. Unlike a data base, VSAM does not provide automatic logging, backout, or recovery techniques upon detection of an error. These must be user implemented.

Thus the philosphy is one of warning. You can still use IEHDASDR to backup volumes if you wish, and you can use SUPERZAP to modify VTOCs and catalogs. Whether these techniques are reasonable in an environment where data is shared concurrently or even simultaneously is another matter.

VSAM also offers a lot of new facilities that if not understood could lead to many mistakes. One example is the ability to process an index component without it's corresponding data component; another is the

degree of dynamic space acquisition permitted in VSAM, a concept totally foreign to ISAM users. Thus VSAM facilities must be evaluated and the proper backup put in place by you, the user, to match the new facility. In large, complex installations, you may not realize exactly what VSAM facilities are being utilized for a cluster you are working with. If your installation sets restrictions that you bypass through ignorance or by design, then your installation will have to bear the brunt of the effort to repair the damaged data. Remember, VSAM is an access method, not a data base. If recovery is a terrible problem for you, perhaps you should consider a data base where the recovery techniques have already been worked out. IMS/VS is such a program product offered by IBM.

BASICS.

OPEN flags are in the catalog.

Every VSAM cluster opened for output has a flag set in the catalog at x'9D' for both the data and index components. If you fail to issue the CLOSE macro, these flags are not reset and will produce warning feedback codes on subsequent OPEN processing. (The common feedback code is x'74' but x'60' is also possible.) If a programmer chooses to ignore these warnings, the cluster is opened and if the programmer has opened for output, a subsequent CLOSE will reset the flags thereby masking the error.

The VERIFY command of AMS opens a cluster for output, issues the VERIFY macro, then issues a CLOSE macro. Since the cluster is opened for output, the CLOSE macro will reset the open flags in the catalog. Note that this is not true of the VERIFY macro as it does not issue CLOSE.

Warning errors are considered terminal errors to AMS commands except for VERIFY and REPRO.

Any feedback codes greater than x'80' are considered terminal errors by all AMS commands since feedback codes greater than x'80' leave the ACB unopened. For feedback codes less than or equal to x'80', the ACB is opened. Thus these feedback codes constitute warnings (except x'00' of course), and normally a programmer might be tempted to continue anyway. For most AMS command, VSAM will terminate because a programmer might be employing dynamic linkage to IDCAMS. This would represent an integrity exposure in that a VSAM command used in this manner might destroy a cluster because it's execution was 'forced' by the user. The only exceptions are VERIFY (already described) and REPRO when processing the input data set.

A system initiated close function is not the same as the CLOSE macro.

During abnormal termination, the system executes various resource termination routines to clean up control blocks and other areas. These routines do not write back buffers marked for update because the abnormal condition might have affected the data in the buffers. This is true for all access methods, not just VSAM.

In VSAM, the AMDSB (Access Method Data Statistics Block), which contains critical information about the cluster component as well as statistics, is written back to the catalog when CLOSE is executed. If CLOSE is not executed, serious errors could result in later processing if CA splits occured in the program that abended. (Even though the space is correctly allocated, the AMDSB holds information about the location of the highest level of index and the high used RBA. If these fields are not correct, VSAM cannot simply process additional space allocated to the cluster.) Note that CLOSE TYPE=T will rewrite the AMDSB without actually closing your cluster and serious programmers might consider the use of STAE or ESTAE to intercept errors and issue the CLOSE macro when possible.

The VERIFY command can be used to repair a cluster than was being processed and did not CLOSE properly in most, but not all, situations. As stated earlier, the basic processing of the VERIFY command is to OPEN, issue the VERIFY macro, and CLOSE. The OPEN recognizes the new extents if any, the VERIFY macro locates the SEOF (software end-of-file) in both components and consequently updates the AMDSB and related control blocks in main storage, and the CLOSE writes the updated AMDSB back to the catalog. There are two cases where this will not produce a usable cluster.
One is if you were using the SPEED option and the abnormal termination occurred during initial load. In this case, VSAM cannot guarantee that it will find the proper SEOF due to the fact that the SPEED option does not preformat a control area with SEOFs before loading it. The recommendation is still to use SPEED as the recovery in this case is quite easy - reload the cluster.
The second case is more serious. It concerns an abend in the middle of a CA split. This could occur via an operator cancel with the force option, power failure, DASD failure, etc. This represents a data lost condition and no amount of VERIFYing will do anything about it.

Note that a feedback code of x'04' is returned to a programmer doing an insert (with return code of x'00' in Reg. 15) to indicate that additional space was allocated to the cluster. Such a time would be a good time to issue CLOSE TYPE=T so as to update the AMDSB more frequently. This would minimize the data loss even in the most serious abnormal termination.
As a final comment here, you use use the VERIFY command for what it was intended - resetting the SEOFs - not as a replacement for other backup techniques. VERIFY does not read your whole data set and 'verify' that all your records are in it. If you want that kind of VERIFY, you will have to code it yourself.

Timestamps. (Location.)

FMT1 DSCB.
This timestamp is the creation time of the space or unique object. It is not used by VSAM for any type of checking and will not be treated further. The new data management SU for MVS (SU 60) date stamps the time of last usage of a data set in the FMT1 DSCB but it does not apply to VSAM objects as VSAM uses it's own data set timestamps in the catalog for this purpose.

Data set creation timestamps. (Recoverable Catalogs only.)
At offset x'05' in the Data Set Directory set of fields in the Volume catalog record. It's field name is DSCRETS, it is 4 bytes, and it contains the high order four bytes of the TOD clock at the time the data set was defined.
At offset x'16' in each data or index record, this same timestamp appears in a field named CRADITS. If the catalog is defined as recoverable, CRADITS will be copied into the data and index records in the CRA. This timestamp can then be used to check multivolume VSAM objects.

Data set update timestamp.
At offset x'30' in the data and index catalog records' AMDSB set of fields, the field is named AMDSTSP and contains the full TOD timestamp in 8 bytes. It is sometimes referred to as the 'system timestamp'.

CRA timestamp.
In the CRA itself, this timestamp is at offset x'65' in the first CI in the CRA (CI 0), and is a 3 byte field in the format YDD.
In the catalog, the timestamp appears as the field CRACRETS at offset x'12' of a record that has a counterpart in the CRA. Since CRACRETS is in a catalog record that is copied to a CRA, the timestamp will appear in all such records copied to the CRA. Thus the CRA itself, the records in the CRA, and the records in the catalog all should have the same timestamp.

Volume timestamp. (FMT4 DSCB.)
This timestamp is in the FMT4 DSCB in two different locations:

x'4C' Considered the 'old' volume timestamp.
x'57' Considered the 'new' volume timestamp.

Both of the timestamps are maintained by VSAM but only the 'new' timestamp is checked by enhanced VSAM. The 'new' timestamp is compared to the timestamp in the volume record in the catalog at offset x'5D' to insure that a downlevel volume is not being mounted.
The exact description of this timestamp in the volume catalog record states: "Volume timestamp, which indicates when the first VSAM data space was defined on this volume." This may lead you to believe that the timestamp is never changed; such is not the case. We will see later that the timestamp is updated when space changes are made on the volume; as long as the volume record in the catalog is also updated at the same time, there will be no inconsistency.

Timestamps - Updating and Checking.

FMT1 DSCB
Generated at creation time, it is used by VSAM as an indicator only.

Data Set Creation Timestamp.
In the Data Set Directory set of fields in the catalog and the CRA records for the cluster, this timestamp is never changed. The purpose is to allow the commands like EXPORTRA and RESETCAT to validate that all volumes of a multivolume data set are at the same level. Since such commands use only the CRA to obtain information, there must be some method of determining that a partial restore of a multivolume data set has not occurred.

Data Set Update Timestamp.
Since this timestamp is in the AMDSB, the only potential time it could be updated is when the AMDSB is written back to the catalog; that occurs only when a CLOSE or CLOSE TYPE=T macro is executed. Not every field in the AMDSB is updated on every CLOSE however. With respect to timestamps, the AMDSB is updated when the subparameter 'OUT' appears in the MACRF keyword in the ACB and the ACB is properly opened and closed.
Since a programmer can process a component of a cluster rather than the whole cluster, it is possible to update one component of a KSDS without the other. Later processing of this same KSDS could cause errors. However, in this case, the data set update timestamps for each component will be different, and the programmer will get a warning feedback code (x'6C') at OPEN. The programmer can choose to ignore the warning; in that case the ACB will be open and a subsequent proper CLOSE will set both components' data set update timestamps equal. All trace of the error will then be removed.

If you have detected an out-of-synch condition between the catalog and an owned volume due to a timestamp being mismatched, it is important that you cease trying to process clusters that are in question. Execution of VSAM commands against a catalog with known faults causes unpredictable results, even with seemingly innocent commands like PRINT. This could lead to data set update timestamps appearing that are considerably later in time than the time of the error, yielding the false impression that the catalog was 'O.K. up to xxxxxxxx.'.

Another error in this area is not knowing that some products open all data set with MACRF=(OUT) even though you specify only retrieval. IMS/VS is one such product. If you 'play around' with IMS and specify cluster components to IMS instead of whole clusters, you could get a data set update timestamp mismatch. This has happened in several cases already, almost always by system programmers looking for a 'quick way out'.

Once a catalog is in error and you continue to try to use it, the results of even error recovery procedures built into VSAM are unpredicable. Thus you cannot assume that VSAM error recovery will 'automatically' fix this timestamp. However, a lot of novice programmers seem to feel just this way - a sort of 'the system did it, let the system fix it' attitude. This is especially true in COBOL programming where previously the system abended if a file didn't open properly. Thus a novice COBOL programmer could introduce still more errors (and they do) by ignoring all warnings and processing anything - absolutely anything!
The VERIFY command will 'fix' these data set update timestamps because it opens for output. If you process a KSDS component, why not execute the VERIFY command as a last step in your procedure so as to eliminate the warning feedback codes in subsequent processing. This presumes you are allowed to process the component in the first place.

CRA Timestamps.
Catalog management insures that all updates to entries in a CRA are at the same timestamp level as entries in the catalog itself. This is to prevent 'updating' a CRA that is downlevel as a result of a restore operation. (If you restored, you were supposed to have run RESETCAT to reset the catalog.) Further, during processing of the CRA itself, such as with EXPORTRA and RESETCAT, the individual CRA records are checked against the timestamp in the CRA's self-defining records to insure that part of the CRA is not out-of-synch with itself.
The basic CRA timestamp is created when the CRA is created and are propogated as new entries are made in the catalog/CRA. They are never updated except when RESETCAT is executed. RESETCAT does not update the CRA timestamps - it updates the catalog to agree with the CRA.

Note Well!
If a downlevel volume containing a CRA is restored and you forget to issue RESETCAT, further processing of the downlevel volume will cause catalog management to recognize the timestamp mismatch of the CRA timestamp with that of the timestamp contained in the catalog records at offset x'12'. When this occurs, the catalog is updated as required by the operation (DEFINE, DELETE, etc.) but the CRA is not updated and NO WARNING MESSAGES ARE GIVEN!
If you subsequently RESETCAT, data changes that occurred before the RESETCAT but after the restore will be lost because RESETCAT forces the catalog to agree with the CRA.

Volume Timestamps.
The purpose of the volume timestamp is to guard against the mounting of downlevel volumes. If a downlevel volume were to be 'accepted' by VSAM, the AMDSBs and/or space maps in the volume records in the catalog could have been changed in the interim. In the case of the space maps, both DADSM and VSAM could have joint ownership of the contested tracks, and both could allocate the same track to their data sets! In the case of the AMDSB, whole extents of a cluster could 'disappear'; actually, the catalog might correctly indicate the extents, but the logic routines would be unable to correctly access the data. (Actually, results are unpredictable.) My point here is you simply can't continue to process. You must get that volume up to the proper level or RESETCAT if it's a recoverable catalog. Anything else invites disaster.

Of interest here is what timestamp is updated by what, and when. First, the 'old' timestamp is no longer used by VSAM but it is maintained for compatability. (It's the one at x'4C'.) This 'old' timestamp IS updated by nonVSAM utilities, specifically IEHDASDR. IEHDASDR updates the 'old' timestamp when it restores a volume. The time placed in x'4C' is the time the dump was taken, not the time of the restore. The implication of updating this timestamp is given on the next page under the discussion of the 'old' rules of VSAM timestamp checking.

Certain rules are observed in enhanced VSAM regardless of whether the catalog is recoverable or not:

- The FMT4 timestamps are NEVER updated by any programmer action whether the program terminates successfully or not. Thus, unlike data set update timestamps, the programmer cannot 'fix' volume timestamp mismatches by ignoring feedback codes.
- In enhanced VSAM, the NEW FMT4 timestamp is ALWAYS compared to the timestamp in the catalog's volume record when the volume is mounted.

The results of the compare for enhanced catalogs is as follows:

NEW T/S = CAT T/S	Open is successful.
NEW T/S ≠ CAT T/S	Feedback code x'68' at OPEN.
NEW T/S = 0	Revert to 'old' (unenhanced) rules.

The old rules were:

OLD T/S GE CAT T/S	Open is successful.
OLD T/S LT CAT T/S	Feedback code x'68' at OPEN.
CAT T/S = 0	Special case. Update the catalog's volume T/S from the value in the FMT4 DSCB at offset x'4C' - the 'old' timestamp.

Notice that for unenhanced VSAM, the system will accept an 'uplevel' volume. The normal way a volume becomes uplevel in todays environment is for the catalog to become downlevel through a restore of a previous version of the catalog. However, recalling the discussion of IEHDASDR on the previous page, it would also be possible to get an uplevel volume in the unenhanced version of VSAM by simply dumping and restoring the volume via IEHDASDR.
Now by SUPERZAPing the new timestamp to zero, you could force enhanced VSAM to revert to unenhanced rules. If the old timestamp was greater than or equal to the catalog's timestamp, you could get VSAM to accept the uplevel volume, even though the catalog's records did not match the space allocation on the volume.
My point here is NOT for you to try this. Rather, I want to emphatically answer the question of: "What harm is there in doing this, especially if it's a catalog that is not recoverable?" Well, the danger is obviously that the space map in the catalog's volume records may not match the space allocated on the volume. Over and over I have listened to people tell me how bad VSAM is because they tried SUPERZAP and it 'didn't work out right'. It is not VSAM's fault if you SUPERZAP some record and you don't know what you are doing. Further, it isn't VSAM's fault if because of the bad SUPERZAP, you have to do a lot of extra work to make the cluster and/or catalog usable again. Let's face it; it's your fault.
JUST BECAUSE YOU KNOW HOW TO SUPERZAP A VTOC, DON'T ASSUME YOU KNOW HOW TO SUPERZAP A CATALOG OR A VSAM TIMESTAMP OR A VSAM FMT1 DSCB!

The next question is when these timestamps are updated. It's an important question for users of non-recoverable catalogs because such users have no other way of checking for volume mismatches. The recoverable catalog user has timestamps in the CRA and duplicate records in the CRA, but the only thing available to the nonrecoverable catalog user is that volume timestamp. The rules for updating volume timestamps in enhanced VSAM are:

Recoverable catalog.	Update the FMT4 'new' timestamp whenever the catalog's volume record is modified, but only do it once after the CRA is opened. This is done whether DADSM is used to physically allocate more space or VSAM suballocation occurs.
Nonrecoverable catalogs.	Update the FMT4 'new' timestamp only if DADSM is involved.

At first it would seem that nonrecoverable is the way to go for catalogs because the idea of having to SUPERZAP less is tempting to many system programmers. I will admit that for certain cases, such as the master catalog in an MVS system (which theoretically should contain only the minimum number of entries), a nonrecoverable catalog might be a good choice. But you should NEVER pick nonrecoverable over recoverable because you think SUPERZAP will be easier. If you say that, you are admitting that you have no real plan for recovery of your data.
I must have been asked a thousand times now: "But with SUPERZAP, at least I could read (maybe) my bad data on my questionable volume, couldn't I?" Yes, you could. But tell me, what are you going to do with bad data? If you think bad data is valuable, perhaps you ought not bother to read the rest of this paper because it's concerned with getting you good data.

Now by this time, you should be convinced that even the presence of good timestamps does not in itself guarantee integrity because a programmer could use IEHDASDR to restore a volume which contains both a catalog and a cluster, losing any changes made since the last dump of the volume. Further, you could restore a volume which contains just a catalog and the timestamps might just match, but VSAM suballocations might have been made since the last dump of the catalog volume which would be lost in the restore. If you used recoverable catalogs, you could obviously use VSAM utilities like RESETCAT to help you here, but if you have nonrecoverable catalogs, it's your responsibility to synchronize the data with the catalog.
Since the major culprit here is IEHDASDR, let's examine IEHDASDR by itself.

IEHDASDR.

First, IEHDASDR has been updated to check the timestamp in the catalog's volume record before it restores a volume, and it will refuse to restore a downlevel volume. This means that if the current backup is either lost or defective, a prior backup cannot be restored unless you either SUPERZAP the timestamp in the catalog (to zero) or 'update' the timestamp on the backup. There is now a standard patch for IEHDASDR that eliminates this check. Some users keep two copies of IEHDASDR, one that does NOT have the patch and is named IEHDASDR and a second with the patch that is renamed so as to be unavailable to most users. This seems quite reasonable. THE PRESUMPTION IS THAT IF YOU USE THE SPECIAL VERSION, YOU WILL COMPLETE THE SYNCHRONIZATION OF THE CATALOG WITH IT'S OWNED VOLUMES BY EXECUTING THE CORRECT UTILITIES.

If IEHDASDR is used to restore a volume that has a catalog on it, there is no timestamp check since the catalog owns it's own space. This is the most frequent use of IEHDASDR in a VSAM environment. As discussed at the top of this page however, even this simple use can lead to trouble if the wrong backup is used. In any event, this use of IEHDASDR causes the data to be reset, not repaired. The data is reset to some level that existed in the past; you are responsible for repairing the reset data so it is current. Did you save all those transactions?

Many VSAM users forget that if they restore a whole volume, they may restore some nonVSAM data sets as well. Many users find that in considering their VSAM recovery needs, their older nonVSAM recovery procedures are inadaquate. This is especially true when moving to DASD devices with significantly higher capacities such as converting from 3340 to 3350 type devices. In the new environment, VSAM might not occupy a full volume. Note that the program product version of IEHDASDR is selective, i.e., it can backup parts of volumes.

If we now consider multivolume data sets, IEHDASDR becomes harder to justify, especially in the nonrecoverable catalog environment. Obviously, backups must be taken of all affected volumes if a restore is ever to be synchronized with the catalog. Now if the catalog is nonrecoverable, the

backups need be taken only if DADSM is invoked since that is the only time the volume timestamps are changed. (You might decide to take backups more frequently for other reasons.) You can limit the number of DADSM invocations by disallowing secondary allocations. This does not solve the problem of VSAM suballocation, but since a programmer is informed of that (feedback code of x'04' paired with a return code of x'00'), the programmer could set a return code that would cause a step to be executed conditionally. Such a step might automatically perform the required backup procedure.

By now you should be thinking of how valuable the RESETCAT command would be in situations where you were forced to reset a volume. Since volume timestamps are potentially updated more frequently when using recoverable catalogs (potentially, once per open of the CRA), the idea of using only IEHDASDR to backup such a volume is less viable. This case is frequently encountered in operations departments where volumes are backed up using IEHDASDR 'for good measure' regardless of other measures taken. It is hard to convince someone not trained in VSAM that the technique used for the last 20 years might have some shortcomings.

One more thing about IEHDASDR. Recall that the password protect bit is set on in all VSAM FMT1 DSCBs, regardless of whether the clusters within the space are password protected or not. Unless the volume being dumped is owned by the master catalog, you will need either a JOBCAT or STEPCAT DD statement allocating the correct owning catalog. This is true even in MVS which would normally dynamically allocate the required volume. While IEHDASDR could be programmed to locate the correct catalog, to do so would represent an integrity exposure because if IEHDASDR was so programmed, a programmer could dump a volume contained password protected entries, restore the volume on a different system using a nonpassword protected catalog, and process the data - all without knowing the password! (Data set passwords are not checked if the catalog controlling that component is not also password protected.)
The JOBCAT/STEPCAT requirement might appear trivial to you, but it isn't trivial if a JCL validation exit makes every occurance of JOBCAT/STEPCAT a comment statement unless confidential accounting controls are present. Not a few customers use this technique today.

A Digression.

Many people looking at VSAM for the first time view all of the above as incredibly complex and difficult to use. Now I don't think any system written by human beings is 100% perfect, before you shortchange VSAM, think of what is is offering. VSAM is a complex data management technique that has three different organizations within it plus alternate indicies and a massive array of options. I think the average person seeing VSAM for the first time tries to imagine a recovery technique that 'work' in all cases regardless of options chosen. Such a person misses the objective; the objective is to design a VSAM cluster to fit your needs, then a recovery technique for the cluster. The objective might be satisfied by a single candidate volume where just about any recovery technique will be successful. The objective might also be a single KSDS with 100 alternate indicies all in the upgrade set. If that's the case, I think it's safe to say your recovery is going to be exponentially more difficult.
By the way, remember that the design of the cluster can take recovery into consideration. There is nothing wrong with designing a cluster, considering recovery, rejecting the original design and substituting a second, etc.

Frankly, I'm quite proud that IBM decided to publish VSAM manuals that speak quite plainly about potential problems if you fail to set up recovery techniques. I'm proud because I like plain truth. It's the only way to conduct business. Would you want it any other way?

VOLUME OWNERSHIP

The VSAM volume ownership rules are well known:

- One, and only one, catalog can own VSAM space on a volume.
- All VSAM space on a given volume MUST be owned by the same catalog.
- Ownership of a volume is not released until the flag in the FMT4 DSCB on the volume in question is removed. Normally, this is accomplished by:
  Using the VSAM DELETE to delete all VSAM objects on the volume and then,
  Using the VSAM DELETE to delete all VSAM space on the volume.

If you lose a catalog and have no backup, you cannot simply define a new catalog and 'recatalog' the existing objects on volumes formerly owned by the lost catalog because VSAM checks the new catalog for volume records that have the same serial number as the volume you are trying to recatalog. The checking is triggered by the presence of the VSAM ownership flag in the FMT4 DSCB. VSAM refuses to update the new catalog because it assumes that some other catalog still owns the volume. (If it was not so owned, the volume ownership flag would have been reset.) Thus, your old volume is unusable with respect to VSAM. You can force release of of the ownership flag in the FMT4 DSCB by executing:

ALTER REMOVEVOLUMES

This command, when properly coded, will remove the ownership flag in the FMT4 DSCB without updating the corresponding catalog. It also overwrites the released VSAM space with binary zeros, so your data is gone forever. The purpose of ALTER REMOVEVOLUMES is to handle just the situation I described - total loss of the owning catalog - where the user wants to simply start over, i.e., the data is backed up by data set and you want to simply restore the data sets via a VSAM utility like REPRO. To issue the ALTER REMOVEVOLUMES command so as to release VSAM volume ownership you must:

- Code the master password of the master catalog.
- Process one, and only one, volume at a time, i.e., only one volume can be named in the ALTER command.
- NOT attempt to remove the master catalog, or any volume owned by the master catalog. (This is due to the fact that in MVS, data sets required for normal operation are required to be cataloged in the master catalog.)
- The FILE parameter is required, even in MVS. It must name a DD statement which, in turn, names the volume which holds the CRA for the volume you are removing - even if it's the same volume.

Recalling that all VSAM commands are also TSO commands, a TSO user, operating in a system with no passwords, could inadvertantly destroy valuable data. Since ALTER REMOVEVOLUMES overwrites the released space with binary zeros, there is no chance of later 'recovering' the data via IMASPZAP. While I admit the idea of allowing everyone to know the master password of the master catalog sounds farfetched to a large user, it is not at all uncommon in a smaller shop - say a recently converted DOS user. In the haste to 'get the conversion done' and with the idea that 'we will not really being using much TSO at first so passwords can wait' which I find to be altogether to prevalent, this situation can easily occur. I hope that by reading this, it will never occur in your shop.

By this time, you must be wondering why you just can't RESETCAT into some other catalog and forget all about ALTER REMOVEVOLUMES. Well, you can do that if:

- The original catalog was recoverable, i.e., you need a CRA.
- The new catalog has the same name as the old catalog.

About that name requirement. Too many people think that you can simply rename things in VSAM like you could with IEHPROGM. Remembering that the name of the catalog is carried in the CRA self-

defining records for recoverable catalogs, in control blocks in storage for open catalogs, and in a special member of SYS1.NUCLEUS for the MVS master catalog. Since changing a catalog name would lead to such complexity, it is not allowed. Thus, to use RESETCAT in this situation, you need a catalog with the same name. If you don't have one with the required name, DEFINE one instead of trying to rename. If you already have one with the same name and it's recoverable, use it. If you have one with the same name and it's not recoverable (the only way this could happen is if you had two separate systems or if you incorrectly defined a new catalog), you will have to delete it and redefine it properly.

The reverse situation, loss of the volume but not of the catalog owning the volume, is also affected by volume ownership rules. To delete VSAM space in the normal manner, it must be empty. If the objects in the space were defined with the ERASE attribute, VSAM must mount the volume to overwrite the tracks in question with binary zeros when the object is deleted; to delete an object defined with the UNIQUE attribute always requires the FMT5 DSCB chain to be updated so VSAM would have to mount such a volume. This would also be true if the last object in VSAM space was deleted and then you deleted the space. The point is that if you try to simply delete all the entries in a catalog for a particular volume, VSAM eventually gets around to mounting the volume. If you can't mount the volume because it doesn't exist, you have a problem.

To handle this problem in MVS, there is a DELETE NOSCRATCH option which allows you to delete all the entries in a particular VSAM space and then delete the space itself without mounting the volume, i.e., the deletion occurs only in the catalog. You cannot use this parameter of DELETE with recoverable catalogs; to handle recoverable catalogs you have the LISTCRA, EXPORTRA, IMPORTRA, and RESETCAT set of commands.
This option is not allowed in either VS1 or SVS. This means that in these systems you may be faced with ALTER REMOVEVOLUMES for the volume that contains the catalog as well as for other volumes involved if some volume owned by the catalog becomes totally inaccessable. In this situation, I would try SUPERZAP on the catalog to remove all trace of the inaccessable volume. There is now an FDP called VSAMZAP that helps in this situation. See your local IBM representative about this FDP.
WARNING!

EVEN WITH THE FDP, SUPERZAP IN A VSAM CATALOG IS COMPLEX. IF YOU MUST USE THIS OR INTEND TO USE IT, PRACTICE ON SOME TEST CATALOG FIRST! IF YOU DON'T, YOU WILL SPEND MORE TIME TRYING TO SUPERZAP THAN YOU WOULD HAVE SPENT SIMPLY DELETING EVERYTHING AND STARTING OVER!

VS1 and SVS allow recoverable catalogs. Give serious thought to using recoverable catalogs in place of SUPERZAP.

If the volume is not totally inaccessable, i.e., the VTOC can be accessed, then it is possible to use the DELETE command in all systems to handle loss of VSAM data on a volume, i.e., partial volume damage. You can specify a DELETE parameter called FORCE which will ignore ERASE definitions and update the FMT5 DSCB chain only. You can use the FORCE subparameter only when deleting space, GDGs, or UCATs. The FORCE subparameter will also cause VSAM to ignore the rule about the space being empty before it can be deleted. If a cluster is in FORCE deleted space, the cluster remains in the catalog but is marked 'not usable'. (It can be deleted later using a normal DELETE. The purpose of this is to inform you that the space you deleted was not empty.) If you force delete a UCAT, all space owned by the catalog on all volumes is immediately released. You cannot force delete a master catalog.

The FORCE option is not allowed for clusters because if it were, it would imply an out-of-synch condition between a catalog and a CRA would also be allowed. The same reasoning applies to the use of the NOSCRATCH option in MVS in a recoverable catalog.
There is one situation involving recoverable catalogs in MVS when the NOSCRATCH option can be coded. The case involves reset of a CRA into the 'wrong' catalog after the catalog has had it's name changed by SUPERZAP. After the reset, (using RESETCAT), you have the situation of two catalogs owning the same volumes. This can be rectified by issuing:

DELETE volser SPACE NOSCRATCH FORCE CAT(old catalog)

Of course this is a violation of VSAM integrity and should never be used except as a last resort, i.e., you know if it doesn't work you are going to have to delete everything and start over. (The command shown releases volume ownership from the 'old' catalog.)

VOLUME OWNERSHIP IN MVS.

It is accepted practice that the following statements represent what should be done in an MVS system:

- The page and swap data sets should not be all on the same volume.
- The master catalog should have the minimum number of entries.
- A data set naming convention should be established.
- The 3350 is a better device than the 3330 or 3340 for page and swap data sets because it has a higher transfer rate and requires less seeking for the same amount of data due to it's capacity per cylinder.

Now all page and swap data sets in MVS are VSAM data sets. They follow the normal data set naming convention of SYS1.xxxxxxxx for system data sets. But all SYS1.xxxxxxxx data sets must be cataloged in the master catalog. Since the page and swap data sets are VSAM data sets, as soon as they are defined, the volume ownership rules force the volumes to be owned by the master catalog. If several 3350 volumes are used to hold the page and swap data sets, then by the rules of volume ownership, all VSAM entities on those volume must also be cataloged in the master catalog.
Depending on the number of volumes, amount of VSAM data, number of page and swap data sets, etc., you may find that you wanted to use the volumes that contain the page and swap data sets for other VSAM clusters that would normally have been cataloged in a user catalog. But because the page and swap data sets are already on the volumes, you are forced to use the master catalog. If your data set naming conventions would have taken you to a different catalog, you may be forced by your conventions to change the data set name.
Of course this situation is not caused by the 3350 units, but it is aggravated by them if you converted from 3330 or 3340 units and wind up with significantly less separate volumes than you had before.
There is no VSAM (or any other kind) parameter that will 'fix' this situation. If this proves to be a problem for you, you may want to restructure your volumes so as to put as much of the nonVSAM load as possible on the volumes that house the page and swap data sets. Another idea is to rethink your data set naming convention; perhaps some VSAM clusters that are not extremely volatile can be renamed and moved to the volumes in question.
In my experience, it has been the renaming mor than anything else that has caused problems; people just don't like to rename data sets. A little prior planning here can go a long way.

REPRO

REPRO manipulates data sets only. However, a catalog is a VSAM data set, so VSAM will allow REPRO to operate on catalogs with some restrictions. In order to understand how REPRO operates on catalogs, you need to know these definitions:

Catalog COPY	MVS only! Both input and output data sets must be catalogs.
Catalog UNLOAD/RELOAD	One of the data sets is not a catalog.

VSAM determines if an object is a catalog by reading CI#2 of the object and checking to see if it's a catalog cluster record (CI #0 is the catalog's data component and CI #1 is the catalogs index component), or, if the CI checked is not #2 (in the case of a directed request), VSAM validates that the catalog record type is U for user catalog. There are restrictions:

COPY — Both objects must be nonrecoverable catalogs. REPRO terminates with no action if either object is a recoverable catalog.

UNLOAD / RELOAD — The new catalog must be on a volume that has the same serial number, name, and device type as the old catalog. Further, the primary space on the new catalog must be large enough to contain the reloaded catalog. Secondary space, if defined, cannot be used for the reload, but it can be used later in normal catalog operations.

Notice that the following are NOT restrictions:

COPY — Will copy nonrecoverable catalogs to dissimiliar devices, from a 3330 to a 3350 for example.

UNLOAD / RELOAD — Can operate on recoverable volumes. However, since REPRO is a simple data set utility, it opens the object as a data set, not a catalog. Thus, the CRA is ignored in the unload/reload situation. If you use REPRO in this mode, you will have to 'fix' this situation yourself. Section III of this paper describes the situation more completely.

Remember, REPRO is really just a simple program that does GETs and PUTs like anyone else. It's speed is therefore materially affected by your BUFND parameter in particular. Recent tests on an MVS system have indicated that optimum speed will require four (4) tracks of the data set to be in memory. If you are using a 2K CI on a 3350, BUFND should be set to 32 as a 3350 has 8 blocks of size 2K per track.

Suppose you now consider REPRO from the standpoint of backing up a simple cluster rather than thinking about catalogs. Suppose you REPRO a cluster to tape and also EXPORT the cluster to tape. Let's compare the two situations.

EXPORT copies both catalog records for the cluster and the cluster to the tape. For a KSDS cluster, the first 8 records contain catalog information; for ESDS and RRDS clusters, the first 6 records contain catalog information.

REPRO operates on data only so the tape will not have any catalog information with the data.

IMPORT of the EXPORTed tape replaces the original cluster if it exists. No merge action is possible.

REPRO allows both replace and merge.

If you have only the EXPORTed tape and desire to merge records instead of replace, you can still use REPRO. Use the SKIP subcommand to skip the proper number of catalog records.

IMPORT will do an automatic DEFINE if it finds the cluster is not defined in the catalog. In doing so, it will use the space attributes in the catalog records on the tape (EXPORTed version) even if the device type is different, i.e., space is NOT adjusted to fit the device. If the EXPORT was done from a 3330 and the IMPORT to a 3350, you will allocate about 2.5 times the required space. IMPORT will not do the automatic DEFINE if the cluster already is defined - it replaces it, but uses the space you specified in the DEFINE.

REPRO has no capability for automatic DEFINE. It always uses the 'right' amount of space because you always specify the correct amount in your DEFINE.

Both IMPORT and REPRO reorganize any data moved. If you loaded your cluster originally by doing partial loads with ALTERs interspersed so as to adjust freespace in a KSDS, REPRO and IMPORT will use just one freespace definition - the current one. Of course, if you use REPRO, you can accomplish the same thing by using the SKIP subparameter on successive REPROs. This is another reason you might want to REPRO and EXPORTed version of the data instead of IMPORTing it.

The merge action of REPRO is especially attractive if you find you must restore a volume for nonVSAM reasons. If the restored volume contains VSAM clusters, and you have backups of these clusters, you can use REPRO with the REPLACE subparameter to 'update' your clusters. The program product version of IEHDASDR which handles partial volume dump/restores should gratly lessen the chance of this happening.

The CRA.

The CRA is actually a small catalog. It has it's own self-defining records and a copy of every record from the catalog that pertains to objects housed on it's volume. If the CRA is on a volume that contains a catalog, then it also contains copies of nonVSAM, alias, path, and GDG records.
The copies are not truly 'exact' because the copy in the CRA holds the CI number of it's corresponding CI in the catalog and vice versa. The importance of this will become apparent later. When executing any command that uses the CRA (LISTCRA, EXPORTRA, IMPORTRA, or RESETCAT), you must remember that the command uses the CRA as the basis for it's action. Even though you 'know' there is 'more' in the catalog, the CRA commands will 'see' only what is defined in the CRA. One effect of this is that the RESETCAT command will delete anything that doesn't match the CRA records. Let's examine some of the checking that occurs for a command using the CRA:

Suppose you assume there are 3 clusters on a volume. You restore the catalog controlling the volume and assume the catalog contains 3 cluster entries for the volume, but you run RESETCAT to be sure.

1. If the catalog has only 1 cluster entry, a data set mismatch will occur because RESETCAT sees 3 entries in the CRA.
2. If the catalog has 4 cluster entries, RESETCAT will determine from space map comparisons that the fourth cluster is no longer on the volume. RESETCAT will delete the erroneous cluster records from the catalog.

In the above examples, if you used LISTCRA instead of RESETCAT, you would get exact messages from case 1 but only a message indicating that something 'other' does not compare correctly from case 2. Again, this shows that the CRA is used in these commands as the basis for comparison, not the catalog.

Space map calculations are handled in much the same manner as indicated above. Suppose, in case 1 above, RESETCAT has already reset cluster A, but finds that cluster B overlaps A. RESETCAT will delete cluster B. Technically, this should never occur, but if you restore a downlevel copy of the volume which contains the CRA, or if I/O errors occurred, it could happen.

Another check involves a field called SUMTT in the data set directory portion of the volume record. This field contains the sum of the starting track addresses for all extents on a particular volume for a given cluster. SUMTT is a kind of cyclic check field - it doesn't represent the sum of the number of tracks allocated or unallocated to a particular object.

The value of SUMTT is predicated on the checking of the data set creation timestamps. If the data set creation timestamps don't match, then the volume is downlevel and the cluster is marked 'not usable' in the catalog. The most frequent cause of this is a restore of one volume of a multi-volume data set. Note that SUMTT is not used if the timestamps don't match.
If the timestamps match, SUMTT is recomputed and compared to the value in the data set directory portion of the volume record. If SUMTT is not equal, the cluster is again marked 'not usable'. This must be done even with the timestamps equal because there must be an error somewhere in the space maps for SUMTT to be unequal. If uncorrected, overwrites of components could occur.

EXPORTRA vs. EXPORT and other miscellaneous information.

When compared to EXPORT, the EXPORTRA command has the following advantages. But it should be obvious to the reader by now that EXPORTRA requires a recoverable catalog while EXPORT does not.

- EXPORTRA can handle a whole volume as well as individual data sets.
- EXPORTRA will export nonVSAM entries and their aliases.
- EXPORTRA will export UCAT connectors and their aliases. This can occur only when using EXPORTRA on a volume that contains a master catalog.
- EXPORTRA will export empty clusters. It should be noted that while EXPORTRA will export empty clusters, the IMPORTRA of an empty cluster may cause difficulties. For example, you can't build an alternate index over an empty cluster.
- EXPORTRA will export GDGs.
- EXPORTRA will NOT export unusable clusters, but it will issue a message for each unusable cluster it encounters. (EXPORTRA used to export unusable clusters, but an APAR changed that. Since the situation can occur only after a RESETCAT, - remember, you are operating on the CRA, not the catalog - a message is just as good. You can't use an unusable cluster - you can only delete it and reset it with a backup.) .

As stated above, EXPORTRA will export a single cluster if you so state in the command. If the single cluster is a multivolume cluster, EXPORTRA will not dynamically allocate (in MVS) any volume. Instead, it will issue message IDC2668I - Data set not exported due to unmounted volume. For this reason, EXPORTRA of multivolume data set should be planned with care. A separate EXPORTRA for each multivolume data set is advised as the parameters necessary on the command can be tricky.

Other useful facts are:

- Page and swap data sets are always bypassed.
- Only one copy of EXPORTRA is ever in execution at one time because EXPORTRA enqueues on an internal name.
- The catalog associated with the CRA is also prevented from being updated when EXPORTRA is running by a similiar enqueue.

At the time of writing, the EXPORTRA portable data set was not interchangeable with that of a simple EXPORT. (Each set of related objects is separated from one another by a special record - x'00008000' - known as a portable data set SEOF.) This situation has been PSRRed and may be changed when you read this.

IMPORTRA vs. IMPORT.

As with EXPORTRA, IMPORTRA is a volume oriented command, as opposed to IMPORT which will import a single data set only. Of course IMPORTRA can import a single cluster if the corresponding EXPORTRA exported a single cluster. You might get the idea that you can select parts of the portable data set from reading the VOLUME subparameter of IMPORTRA. The fact is, you can't. The parameters of OBJECTS are there to allow you to change volume serial numbers and device type as the entry is being imported. All entries in the portable copy are always imported.

If IMPORTRA encounters an empty predefined cluster, it will NOT use it's new space attributes as IMPORT will do. Instead, IMPORTRA deletes the empty cluster and inserts the old one, with the old space definitions. Users with both 3330 and 3350 devices installed should be wary of this difference, especially if you are going to 'backup' a 3350 with multiple 3330s, or plan to use 3330s to backup critical clusters now on 3350s. Consider either generous secondary allocations or a separate backup for such critical clusters.

When considering IMPORTRA processing, the first parameter of the command that might cause some difficulty is the OUTFILE parameter. If you specify it, you must supply a DD statement associated with it, and the DSN parameter on that DD statement will be used by IMPORTRA internally for the purpose of ALTERing data set names. Thus the name you choose cannot exist in either the catalog or the portable data set. In MVS, you can omit the OUTFILE parameter. In that case, IMPORTRA will use dynamic allocation to satisfy the volume requirements, and no renaming will occur.

The processing can be outlined as follows:

- Obtain the cluster information from the portable data set.
- Attempt to DEFINE the cluster.
- If the DEFINE returns a condition code of 8, (meaning it was already in the catalog), DELETE the cluster and reDEFINE it.
- If OUTFILE was specified, rename the cluster component to the name specified on the DD statement associated with OUTFILE. The data and index component names are not changed.
- The cluster is then opened, loaded, and closed in the normal manner.
- The cluster is then renamed to it's original name if OUTFILE was specified.

Note that renaming the data and index components would require invocation of DADSM to change FMT1 DSCBs in the event of UNIQUE allocations. For a user with exacting data set naming conventions, the renaming process allows you to reDEFINE a cluster with the same name as that of a cluster that IMPORTRA was importing when a failure occurred. Such a user, assuming a separate backup is available, could use the separate backup to 'get going' while the IMPORTRA failure is analyzed. To be precise, assume the following:

A KSDS cluster has the names:

MASTER.FILEX.CLUSTER
MASTER.FILEX.DATA
MASTER.FILEX.INDEX

The cluster is properly exported using EXPORTRA and is now in the process of being imported with IMPORTRA when IMPORTRA fails without renaming the cluster back to it's original name, i.e., OUTFILE was specified, using a DSN of IDIOT.DATA.SET. The situation in the catalog is now:

IDIOT.DATA.SET	cluster name of your cluster that was MASTER.FILEX.CLUSTER
MASTER.FILEX.DATA	data component - name not changed
MASTER.FILEX.INDEX	index component - name not changed

If you now attempt to DELETE the cluster MASTER.FILEX.CLUSTER, you will get a 'data set not found'. If you attempt to explicitly DEFINE the cluster MASTER.FILEX.CLUSTER with the data and index component names of MASTER.FILEX.DATA and MASTER.FILEX.INDEX respectively, you will get a 'duplicate data set' message caused by the data and index component names still in the catalog. If you try to reissue IMPORTRA, IMPORTRA will fail to import MASTER.FILEX.CLUSTER because it receives internally the same messages just described.

Notice that you can explicitly DEFINE the cluster MASTER.FILEX.CLUSTER with different data and index component names. As stated earlier, this will allow you to 'get going' when dealing with an exceptionally critical situation. THIS SHOULD NOT BE YOUR NORMAL COURSE OF ACTION AS IT COMPRIMISES YOUR OWN DATA SET NAMING CONVENTIONS.

You can DELETE the cluster IDIOT.DATA.SET which will delete all three names. This would then allow you to rerun IMPORTRA or DEFINE your cluster in the normal way and run a separate backup. In the case of continued IMPORTRA failure, some customers have written their own utility which will read an EXPORTRA portable data set and extract the necessary data to rebuild a single data set.

You might ask how, in a well organized shop, you can be left with a failing IMPORTRA as your only backup to a critical data set? Indeed, how can IMPORTRA fail in the first place - wasn't it checked out? If the catalog is that bad, isn't there a procedure for a damaged catalog?

Questions like these are the point of this whole paper. I visited one shop that was having VSAM 'problems' only to find they were YEARS behind in maintenance; in another, the sole VSAM trained individual had just left. It seems no one else knew anything about VSAM. These shops were large shops, with multiple CPUs. You would think they would know better. Yet even today, some managers believe VSAM is nothing more than a new way to code ISAM. If your manager feels that way, perhaps you can point out why VSAM needs to be understood; at least you can say that the VSAM utilities are part of VSAM and that the VSAM maintenance affects them (or may affect them). At the very least, you should have the ability to test the backup procedures when new VSAM maintenance is applied.

RESETCAT.

RESETCAT is a powerful command, but you should not assign it properties it does not have. For example, RESETCAT will NEVER:

- Recover a VSAM data set, or any other kind of data set. RESETCAT only restore accessability to what is already there.
- Correct any permanent error in the catalog or CRA. RESETCAT will try to do secondary allocation and logically delete what was on a defective track, but RESETCAT does not use the ATLAS macro in an attempt to do dynamic alternate track assignment.
- Read the HKR hunting for errors. RESETCAT will use the HKR to do normal inserts, deletes, etc., and will find out if there are errors in the area used, but RESETCAT does not check the entire HKR nor will it try to do any dynamic alternate track assignment if it does find an error in the HKR.
- Attempt to process a catalog that cannot be opened. This means the catalog's self-defining records must be intact.
- Attempt to process a catalog in which an insert into the index set cannot be done.

From reading the above, it may appear that a catalog must have secondary allocation coded in it's DEFINE for RESETCAT to execute. That is not true. However, you do need room to perform inserts, so the prudent systems programmer will periodically check the extents of catalogs for signs of a catalog full condition. If the catalog is full and you have no secondary allocation left (or you didn't code any secondary allocation, the catalog must be recovered into a larger extent before RESETCAT will be able to perform it's required inserts. This means you may not be able to successfully execute RESETCAT if your catalog is full. (You may be successful - it depends on what RESETCAT has to do.)

RESETCAT requires the master password of MCAT to operate.

If RESETCAT encounters an error in the HKR, it renames the entry it is currently processing, gives you a message, and continues processing. You are left with both names in the catalog. Later, you can examine the messages and take your own action. (You might be able to ALTER one of the entries to the correct name and delete the other, as an example.) This means that it is still possible to have a defective catalog after running RESETCAT! You may then want to reexecute your recovery procedure against an earlier version of the catalog, probably restored by IEHDASDR. Since RESETCAT can modify the CRAs as well as the catalog itself, it is well that you use IEHDASD to back up the complete catalog system prior to a major recover attempt, i.e., the catalog and all volumes that will participate in the recovery.

Obviously, RESETCAT cannot reset a defective catalog entry if the corresponding entry in the CRA is also defective. Recalling that the CRA is actually a small catalog of itself, with it's own self-defining records, RESETCAT will fail if it cannot open the CRA, i.e., the CRS's self-defining records are bad.

If the CRA's volume records are damaged, RESETCAT normally terminates, but you can force it to continue by coding the IGNORE subparameter. While IGNORE forces RESETCAT to physically execute, RESETCAT proceeds to mark all records in the catalog associated with the volume upon which the CRA resides as 'notusable'. Remember, there may be other volumes involved with a single execution of RESETCAT so perhaps only one volume will have errors. Now the 'notusable' attribute is set at the data/index level not at the cluster level, so it is possible to have one component of a KSDS 'notusable' while the other component is 'usable'. (Exception: A KSDS cluster with the IMBED option; both components will be either 'usable' or 'notusable'.) Note that in the case of a multivolume cluster, one volume may have caused a SUMTT or data set creation timestamp mismatch; the space on that volume will be freed for suballocation and the component marked 'notusable', but the data on the other volumes is salvageable using EXPORT or EXPORTRA.
If you use the LISTCRA command before you run RESETCAT with the IGNORE option, examine the mismatches, and use EXPORT or EXPORTRA to salvage these data sets, you can save yourself a lot of grief.

If you execute RESETCAT with the IGNORE option and the damage is not in either the self-defining or volume records, then all the entries not in the damaged area will be reset, but those that fall either totally or partially in the damaged area will be deleted from the catalog. Again, it is most prudent to use LISTCRA and EXPORT or EXPORTRA in this situation to salvage what can be salvaged.

What's the point of all this? It's simple. Don't fall into the trap of thinking that RESETCAT can fix anything. Just because you defined recoverable catalogs, you don't throw LISTCRA, EXPORTRA, IMPORTRA, EXPORT, IMPORT, REPRO, and VERIFY out the window. RESETCAT is a command to use with these commands - not in place of them!
The question that's really being asked here is: "What do I do if the CRA is damaged?" Well, if you know it's damaged, but the catalog is O.K., you can use EXPORT, IMPORT, etc. to access the data. Then you can ALTER REMOVEVOLUMES, DELETE, etc. (clean up the volume) and reDEFINE you clusters. Your liability here is quite small.
What if both the CRA and the catalog are damaged? Well, I'm getting ahead of myself - that's in PART III, but basically, the answer is you better have backups of all volumes involved.

RESETCAT cannot reset the CCR (Catalog Control Record). The CCR in a CRA is the CCR of the CRA, not the CCR of the catalog.

RESETCAT allows you to reset a CRA on VOL111 to a catalog on VOL222 where VOL111 has a catalog defined on it, as long as the catalog has the same name as the catalog on VOL111. This feature is provided in case the catalog on VOL111 is totally unusable yet the data on the volume is still intact. When you do this, RESETCAT deletes the catalog on VOL111 and frees it's space for secondary allocation purposes; the space is still VSAM space, i.e., it appears in the space map in the catalog on VOL222. Effectively, ownership of VOL111 has been transferred to VOL222.
NOTE WELL!

There have been APAR fixes against RESETCAT in this situation, specifically with regard to it's handling of nonVSAM, GDG, and UCAT entries, and it's handling of the second qualifier of the name of the space associated with the catalog on VOL111 (changing it from Z9999994 to Z9999992). Be sure you have these fixes on your system!

VSAM BACKUP AND RECOVERY

THE DATA SET - THE VSAM CLUSTER

If you have not read PART I of this paper, read it now!

ASSUMPTION:
At all times, the customer has either a workable backup copy of the cluster which is restorable (reset) and then repairable using some form of journaled transactions or the like, or, the customer can recreate the cluster from 'scratch', such as a cluster of mathematical tables.

Since the most difficult cluster to recover is the KSDS, I will concentrate on it. Recovery of ESDS and RRDS clusters will then fall into the catagory of a special case of KSDS recovery. The important points involved in the recovery of any VSAM cluster are:

In the catalog:

- Extent information.
- Software EOFs by key range.
- AIX and PATH relationships.
- Volume information.
- Passwords.
- Temporary export indicator.
- Timestamps.
- High-used RBA.
- Statistics.
- Location of the highest level index record.
- Key range information.
- Open indicator.
- Page and swap data set flags - MVS only.
- Space map in the volume record.

The cluster itself contains two separate, processable data sets if it's a KSDS cluster:

- Index data set.
- Data data set.

All clusters have a cluster record in the catalog which relates the cluster components.

In the CRA, if the catalog used was defined as recoverable, you have:

A duplicate of the catalog information. This 'duplicate' information is not exact however, because the CRA records relate to the catalog records and vice versa. The information that is not the same is the CI numbers that relate the records - in the catalog, they indicate CRA CIs, in the CRA, they indicate catalog CIs.

1. Suppose the CRA is destroyed.

The first question is how do you know that the CRA is destroyed. You know when you get message IEC3311 028-008 that there has been a CRA error. (IEC3311 028-004 is a catalog error.) VSAM will refuse the operation requested if any CRA error occurs. If you were processing at the time of the error, that is, some one else destroys the CRA while you are processing, you will receive notification when you close the cluster because the statistics cannot be properly updated in the CRA records. If you were not processing, you would obviously not be informed until the next time you tried to process the cluster; for this reason, you should periodically use the LISTCRA command to check the CRA of your cluster.

Once detected, the natural question is: "I know that EXPORTRA and RESETCAT cannot operate on my cluster's CRA records, but will EXPORT or REPRO be refused, especially since EXPORT sets the export indicator?" The answer is, happily, no. EXPORT will end with a message (IDC3351I - VSAM CLOSE ERROR), but the cluster will have been exported and the portable data set will be acceptable to IMPORT. Therefore, you can:

A. Restore the whole volume and run RESETCAT.
This is really not cluster recovery - it's more like volume recovery. If you choose to do this, you must realize you are restoring more than just your own cluster. Therefore, you had better read the section on catalog recovery where I cover this type of recovery.

B. EXPORT your cluster, get some one else to repair the damaged volume, then IMPORT.
The idea behind this is that repair of the damaged volume is probably going to involve ALTER REMOVEVOLUMES and DELETE FORCE, so you get a copy of your cluster before they do all that.

C. REPRO to a different volume using a different cluster name, then use ALTER to rename.
This is the same as 'B' above. REPRO is faster than EXPORT, but if you are using REPRO to go directly to another VSAM cluster, the REPRO and/or the ALTER might fail. (If the CRA is damaged, the catalog might be damaged also.) If that happened, you could be left with no data at all.

Basically, you should look at CRA damage as an early warning of impending problems. The one thing NOT to do is to continue to use the volume with the damaged CRA. Once a catalog or CRA is damaged, even innocent commands like PRINT could potentially cause even more damage because how can you state that OPEN, CLOSE and error recovery modules will 'work' if you start with bad data?

2. Suppose the index is destroyed.

The probable cause of this problem is a failure to 'properly' close the cluster. If an abend occurs in the middle of a CI/CA split, the system does not flush buffers - for VSAM or any other access method. Obviously, the index can be damaged. In a multi-tasking environment, the problem can occur if a subtask abends that has a separate AMBL while other tasks continue to process. (This is the case if you elect NOT to use subtask sharing.) You can use the ESTAE/STAE - ESTAI/STAI interface to intercept these abends and issue the proper CLOSE macro if the buffers appear to be intact.
In a non multi-tasking environment, the same problem can occur if an operator cancels a VSAM job that was in the middle of a split, or if the user elects to do VSAM sharing with another partition, region, or address space using SHR(3,3) and loses write integrity. That is, two or more users update a VSAM cluster simultaneously.

One way of preserving as much integrity as possible in these situations is to instruct programmers to make judicious use of CLOSE TYPE=T. That macro updates the catalog without actually closing the cluster - it's a sort of catalog checkpoint. While it can't guarantee recoverability, it may save you from executing complex recovery procedures and, in some situations, can actually eliminate most of the recovery. (You still need to at least validate that your cluster is O.K.)
I have already mentioned the second way of preserving as much integrity as possible - use of ESTAE/STAE. In MVS, the ESTAE coding has been greatly simplified through the use of the SETRP macro so there isn't any excuse in that system for not considering these macros. Admittedly, in VS1 and SVS the coding of STAE is not quite as clean, but nevertheless, every serious users of VSAM should avail themselves of the protection STAE offers.

If you suspect that integrity has been lost, run VERIFY first. It might not correct the problem, but at least it will set the high-used RBA correctly so that later recovery procedures can process the whole cluster. Note that you will probably need your own program to validate that the cluster is either intact or destroyed - no VSAM command can validate your data. If you find you do have a problem, you can:

A. Import a good copy of the cluster. You might as well DELETE and reDEFINE before you IMPORT as that allows you to change space allocation to what you need now. (Presumably, the backup doesn't reflect all the latest transactions.)

B. Use REPRO from a backup copy. Theoretically, this should offer the chance of saving the latest transactions because REPRO can merge into an existing cluster. However, remember that we got here because the index was destroyed. How can you merge into something that is destroyed?
You could use REPRO into an empty cluster however. That would be similiar to the IMPORT mentioned earlier - the difference is really in how you took the backup.

C. Restore the volume, wholly or partially, via IEHDASDR. Before you consider this move, read about catalog recovery. There are a lot of things to consider in even a partial restore of a volume, and if you decide to restore a whole volume, what are you going to do about the other data sets (nonVSAM) and clusters you restored?

Caution:
If you define a cluster with the REUSE attribute, REPRO, since it opens the target cluster for output, will not do a merge but will instead be building the cluster from scratch, i.e., it will be REPROing into an empty cluster. This may be what you want, but you should also be aware that once the open is done, the target is empty. Any ideas of saving the 'good' records are then worthless.

3. Suppose the pack is defective.

If you think only a track or two went bad, you could use the ATLAS macro to assign an alternate followed by a reset and repair of the cluster that occupied the track. This is not easy, and it requires you to write your own programs to do it, but in the case of a critical on-line application using non-removable DASD, it might be attractive as a first measure.
If the whole pack is defective, you must first clean up the catalog that owned it because if you don't, VSAM volume ownership rules will prevent you from simply using the VSAM commands to place the data on another volume via a backup copy. Clean up means using:

DELETE FORCE and/or DELETE NOSCRATCH (NOSCRATCH is MVS only.)

Obviously, you can restore the whole pack via IEHDASDR. This is attractive IF the whole pack is either one cluster or a collection of easily reset and repaired data sets. In other words, you might be better off in the long run by resetting individual data sets rather than whole volumes. If you say you don't know what was on the volume so you have to restore all of it to preserve what was there, then you are saying you really don't have any cluster recovery procedures at all. In fact, you don't have any recovery for anything, because how can you say you don't know where your master files reside? Remember, VSAM clusters are not temporary clusters - they are permanent master files!

4. Suppose the high-used RBA is in error.

Run the VERIFY command. (The VERIFY macro in a program will not fix this unless you CLOSE the cluster after the macro. The VERIFY macro does not write back to the catalog; it depends on the CLOSE macro for that.)

5. Suppose the timestamps of the data and index components do not match.

Running VERIFY or any program which opens the cluster ignoring the warning feedback codes will 'fix' these timestamps, assuming the CLOSE is proper. Of course, I assume you found out WHY those timestamps were mismatched. If you don't, and later have problems....then you desreve them!

6. Suppose the volume timestamp does not match it's catalog entry.

My advice, if you have a recoverable catalog, is to run:
LISTCRA/EXPORTRA/IMPORTRA or
LISTCRA/RESETCAT
My advice if you don't have a recoverable catalog is to clean up the catalog and volume using DELETE FORCE and ALTER REMOVEVOLUMES respectively, then use DEFINE, IMPORT, REPRO, or your own programs to recover the clusters.

I am aware that SUPERZAP is very tempting here. By SUPERZAPing the FMT4 DSCB to the timestamp in the catalog, you can use the volume right away. 'Use' is probably not the right word - it's more like 'mount' the volume and hope for the best. Technically, what has happened is that the wrong backup has been restored.
If you have a recoverable catalog, you can run RESETCAT to synchronize the volume with the catalog but if you don't have recoverable catalogs, the SUPERZAP is nothing more than taking a chance. You should be aware that chance is not the same as a 100% guarantee and take steps accordingly, i.e., you better have backups of the data on that volume if your SUPERZAP fizzles!

7. Suppose AIX and/or PATH relationships are in error.

LISTC and/or LISTCRA to establish the extent of the damage.
DELETE items known to be in error. If whole data sets must be deleted, then they will have to be reset.
DEFINE the path relationships. If an AIX had to be deleted, it will have to be rebuilt.

Some customers have gone ahead and rebuilt AIX clusters using a different name until the extent of the damage is known and the old AIX deleted. Then, they use ALTER to rename the 'temporary' AIX the it's correct name. This is all possible if you can run for a time using different data set names.

8. Suppose the SEOF (software end-of-file) is in error.

This is really the same as the high-used RBA problem because the SEOF is used to check the high-used RBA. The fix is the same - use the VERIFY command.
Remember that in loading a cluster for the first time with the SPEED option, the CAs are not pre-formatted with SEOFs, so you may not be able to use VERIFY to recover an error in initial load if you choose the SPEED option. (It is wise to take SPEED however because it significantly improves the load time to the point that a reload is probably cheaper than a recovery in most cases.)

9. Suppose passwords are erroneous.

Use ALTER. You will need the master password of either the cluster or the catalog to run ALTER. Since the presumption is that the passwords are erroneous for your cluster, the master password for the catalog is a realistic requirement. When you use the master password, the user security verification routine is bypassed - it has to be; otherwise, you could have a bad password and never get by the USVR to fix it!

10. Suppose any volume, key range, or statistical information (AMDSB) is incorrect.

Realistically, only RESETCAT can be used here. You can use ALTER to add additional volumes and/or key ranges, but you can't use ALTER to repair or reset incorrect volume and/or key range information that already exists.

In all of the above considerations I deliberately left out EXPORT/IMPORT because I assume that your intentions are cluster repair, not reset followed by repair. Obviously, if you have a backup copy of your cluster, you can use it to reset the cluster followed by the necessary repair, i.e., reprocessing the transactions that occurred after the last backup. As explained earlier, either EXPORT or REPRO can be used to handle the backup; IEHDASDR or a similiar utility is also a possibility but EXPORT or REPRO are preferred because they are cluster oriented whereas IEHDASDR is volume oriented. EXPORT and REPRO operate only on a single cluster; it's true that the program product IEHDASDR allows the user to operate on parts of volumes, but with additional control card requirements. Further, what is the practical chance of using IEHDASDR to recover a portion of a VSAM suballocated space? Pretty slim. Why take a chance with IEHDASDR - use EXPORT or REPRO.

If you have recoverable catalogs, you can of course use RESETCAT. Now RESETCAT is a volume oriented command and this section is concerned with clusters only. The question is: "Will you be allowed to use RESETCAT in your installation to recover a single cluster?" The answer to this question should be known to you in advance, i.e., part of the design of your recovery procedures should be an evaluation of what commands will be allowed or disallowed in a given procedure. The point here is that while RESETCAT represents a good choice in general, in particular in requires that no data set in the catalog in question may be open and that the catalog must be able to be brought under exclusive control of RESETCAT.
In MVS, the master catalog cannot be the object of a RESETCAT while it is in use as a master catalog; this restriction does not apply to VS1 or SVS.
So the question remains. Before you decide that RESETCAT is the answer to your recovery problem, you had better check with others at your account. They might not want to close their clusters while you issue a RESETCAT.

My best recommendation for cluster recovery is to design it into the application. Further, offer application programmers several user catalogs. By doing both of these things, the degree of difficulty of recovery can be predicted in advance, and the programmer can then choose to implement appropiate utilities, macros, and commands in the job stream that will allow full recovery of a cluster within the prescribed time limits. Installations have gone years without the loss of data using such methods.
One installation, heavily committed to structured programming and the chief programmer approach, uses live demos of an application programmers recovery procedure to validate that the procedure is not merely a textbook solution. Once such a recovery procedure passes it's test, it is carefully documented; others then can use the solution without extensive research. Each new usage is still tested however.
This technique has enabled at least one customer to enjoy no data loss at all using VSAM, and with a minimum of time spent in the design of VSAM error recovery as well.

THE USER CATALOG (UCAT).

First, a word about placement. The best place for a UCAT is on a volume totally dedicated to VSAM. The reason is that placement of a UCAT on such a volume will allow you to use the IEHDASDR utility to restore the volume without worrying about nonVSAM data sets. If the catalog is recoverable, you can of course use RESETCAT as well.
Remember, if a multi-volume cluster is concerned, all volumes of the cluster must be recovered, not just some of the volumes. Regardless of utility used, you must be alert to the presence of multi-volume clusters and guarantee that all volumes concerned are recovered.

The second best location for a UCAT is a volume with nonVSAM data sets that change infrequently. Again, the reason is that should you desire to use IEHDASDR, the potential damage to the nonVSAM data sets is reduced. (While it is true that the program product IEHDASDR will restore partial volumes, it requires control cards to do it - and every control card is a potential source of error.)

From a performance standpoint, major workload areas - TSO, production, data base, program test, etc. - should have their own UCAT. Most customers find this to be a natural segregation anyway. My point is that the basic recommendation is to use recoverable UCATs in all major areas. It is true there is some overhead in the use of a recoverable catalog, but unless you are doing literally thousands of DEFINEs, DELETEs, ALTERs, etc. per shift (as in an educational environment), the fact that you have access to LISTCRA, EXPORTRA, IMPORTRA, and RESETCAT is a major advantage that far outweighs the overhead.

NONRECOVERABLE UCATS.

Detection - how do I know I have a bad UCAT?

Normally, you would suspect a bad catalog when:
- Many VSAM objects cannot be OPENed.
- The catalog itself cannot be OPENed.

In the first case, you can do a LISTC to help find the extent of the damage, but in the second case, you better know what was there because it isn't there now! (Did you save the last LISTC?) SMF records can be scanned in the latter case to help you determine what was there - if you systematically save the SMF data. Obviously, a damaged volume falls into the latter case.
It may seem a bit silly to you now to read that a damaged volume can't be used, but think of those RJE networks - will your users at the end of those lines realize that the catalog can't be opened?

'RECOVERY' of the nonrecoverable catalog.

You should not expect to repair the catalog. While there has been some limited success using SUPERZAP, it requires a great deal of knowledge about catalog internals, and if nothing more, SUPERZAP is time consuming. It is usually a lot faster to restore the catalog and then restore and/or repair each of the entries in the catalog. Yes, SUPERZAP is that complicated.

RESTORE (IEHDASDR)

This produces an immediately processable catalog, but all entries in the catalog must be considered down-level, including nonVSAM entries. If the catalog controls multiple VSAM volumes, then:
- VSAM space allocated since the last backup is totally unknown.
- VSAM volumes acquired since the last backup are useless with respect to VSAM due to ownership. rules, i.e., the catalog doesn't own them, yet the ownership indicator is set in the FMT4 DSCB.
- VSAM suballocated space assigned since the last backup is lost.
- VSAM statistics are lost permanently.

After restoring the UCAT, your job has just begun. Consider three cases.

Case 1. A cluster lies wholly in the volume restored.
This cluster can be repaired by reexecuting jobs that processed the cluster subsequent to the last backup, or, if some sort of journal was kept, you can repair the cluster by executing a special program coded by you that processes the journal.

Case 2. A cluster lies wholly on a volume owned by the UCAT, but the volume was acquired after the last backup.
This cluster cannot be accessed because the restored catalog no longer has ownership of the volume and cannot get ownership due to the presence of the ownership flags in the FMT4 DSCB. The general approach here is to use ALTER REMOVEVOLUMES to release volume ownership flags on the volume in question and to redefine space, clusters, etc. and rebuild all VSAM entities on the volume in question.

Case 3. A cluster lies partially on the restored volume.
You must consider the catalog entries for such a cluster invalid. You should delete such a cluster and rebuild from your backup much like case 2 above. The difference is that ALTER will probably not have to be used as ownership is still maintained by the restored catalog

Some further reminders. The timestamp of the volumes must be delt with. Recalling that volume timestamps are updated whenever DADSM is invoked for nonrecoverable catalogs, owned volumes may be uplevel as a result of the restore. As I have stated many times previously, you may want to try SUPERZAP here to 'fix' the timestamps. If you do, there is ABSOLUTELY NO GUARANTEE that the data on the zapped volumes will be processable.
Many customers say: "Well, at least I can then get at some of my data." Probably true. If some of your data is what you want, then I guess you ought to try superzap; if you want all of your data, you better have some good backup.
Other customers have said: "We do such frequent backups that we can just restore everything and run. What's wrong with that?" Nothing - today. But when your data expands to cover many volumes, will you be easily able at that time to implement different recovery procedures? Isn't it better to pretend today that you have that much data so that in the future, your backup procedures are solid - using the acid test of actual experience?

REPRO with Nonrecoverable Catalogs.

REPRO can be used to make 'backups' of a catalog. As stated earlier in Part I, REPRO has two methods of creating these backups:

UNLOAD/RELOAD	Available in all systems, unload/reload will operate on any catalog, but the reload requires a catalog on a volume that has the same name, volume serial, and device type as the original catalog. Unload/reload uses portable media and the backup itself is NOT a catalog.
COPY	Available only in MVS, copy functions only on nonrecoverable catalogs. Copy does not use portable media - it copies directly to another catalog which does not have to reside on the same device type, nor does it have to have the same name or be on a volume with the same serial number. It does have to be empty, and it's primary space must be large enough to contain the copy.

Since the unload/reload function is available in all systems, let's look at it first.

UNLOAD/RELOAD FUNCTION OF REPRO - NONRECOVERABLE CATALOGS ONLY.

The AMS SRL contains a good description of this process, but I will review it here. The unload is simple, requiring only the following:

- A KSDS, ESDS, or SAM data set.
- A STEPCAT DD statement to identify the catalog being unloaded.
- Prevention of catalog update while unloading. (REPRO itself does not prevent other users from being open to a cluster at the same time as the unload. One way to insure this is to obtain exclusive control of the catalog, i.e., code DISP=OLD on the STEPCAT statement.)

The reload is a bit more complicated. You can either reload into a newly defined catalog or into a version of the catalog that already exists. The reload reacts differently in each case.

Newly defined catalog.

A check is made by reload and if the catalog is new, reload bypasses reloading the original version of the volume record. The assumption is that if the catalog is new, nothing else exists with respect to VSAM on the volume, i.e., there are no other data spaces.
This means that the volume information in the original version of the catalog for the volume targeted by the reload is lost. Other volume information is not lost - only the volume information for the volume upon which the reloaded catalog exists. If clusters existed previously in those spaces now lost, they can be accessed, but not extended. To handle this situation, you have to EXPORT PERMANENT to remove the entries from the reloaded catalog, DEFINE sufficient space to receive the clusters, then IMPORT the clusters.

Old version of the catalog.

Reloading here results in a catalog equivalent to the one that existed at the time of the backup. This means that entries that exist in the catalog with the same name are replaced with the information from the unloaded version, entries that do not exist are inserted, and entries that exist only in the catalog and not on the unloaded version are deleted.

Obviously, clusters added since the backup was made are deleted and therefor cannot be accessed. You must have a separate backup of these. Clusters deleted since the last backup will reappear - at least in the catalog. If you try to process such clusters, unpredictable results will occur as the space map in the catalog may not properly reflect VSAM track usage. You correct this situation by reissuing the DELETE. Note that the FMT1 DSCBs for VSAM space added since the last backup will still exist in the VTOC, but will be nonprocessable by the now reloaded catalog since they do not appear in the space map. Such space can be recovered by scratching the FMT1 DSCBs associated with the spaces. If whole volumes were added since the last backup, they too will be unusable. You must use either DELETE with the FORCE option or ALTER REMOVEVOLUMES to release volume ownership and recover the clusters on these volumes using separate backups of the clusters.

If a cluster existed at the time of the unload, but was extended since that time, the new extents will not be recognized. If the extension was into space already allocated at the time of the unload, you can recover these allocations by using the VERIFY command. If the extension was done by acquiring more space however, all data in that space is nonrecoverable. Attempts to process the data in such space will result in a logical error.
For ESDS clusters, the missing records are all at the end of the cluster. If you know which records are missing, you can add them to the end of the cluster.
For KSDS clusters, the problem is much more complex as records may have been moved via a CA split. Your only real alternative here is to recover the entire cluster.
Data in RRDS clusters acts like data in ESDS clusters in this situation.

Remember, you can't recover the space that was added since the last unload. If you add records to the end of your ESDS cluster for example, and there is not enough room, VSAM will go through it's normal allocation procedure to obtain more space - it will not use any of the space added after your last unload because there are no catalog records for it.

Before I go on to the copy facility of REPRO, I want you to realize that the unload/reload function of REPRO DOES NOT MAKE YOUR CATALOG RECOVERABLE! Notice that many of the requirements after the reload are quite similiar to what you have to do after using IEHDASDR. Thus, whether to use REPRO or IEHDASDR is a choice you must make only after examining what will be cataloged in your catalog. If the catalog is rarely changed (such as the master catalog in MVS), REPRO is a good choice for backup; if the catalog is frequently updated, I would recommend you make it recoverable. If you already have nonrecoverable catalogs and have to keep them for a time, then only you can decide which is better - REPRO or IEHDASDR.

COPY FUNCTION OF REPRO - MVS NONRECOVERABLE CATALOGS ONLY.

In order to use the copy function, you should first do the following:

- Make sure AMS is authorized. The copy function requires this.
- Determine the amount of space required to hold the copy. There is no straightforward way of doing this, but the AMS SRL lists methods and formulae that are helpful. The problem is most critical in changing device types. The requirement here is that the primary space be large enough to hold the copy.
- Prepare JOBCAT and/or STEPCAT statements for both the receiving and source catalogs. (Concatenation can be used.) If you are copying the master catalog, only one JOBCAT or STEPCAT statement is necessary - the one for the receiving catalog.
  You must use JOBCAT/STEPCAT statements as VSAM will not use dynamic allocation in this case.
- Define an empty catalog. It must be empty.

You are now ready to perform the actual copy. Here is a suggested sequence:

1. LISTCAT all the UCATs alias entries in the MCAT. The EXPORT that follows will delete these alias entries and a later IMPORT will not restore them. (You restore them by defining them with a DEFINE ALIAS.)
2. REPRO the source catalog into the target.
3. EXPORT the source catalog from the MCAT with the disconnect parameter. This will also remove any alias entries from the MCAT!
4. DELETE the source catalog from the target catalog. This is necessary because REPRO copies the source catalogs self-defining records into the target. This DELETE removes them.

At the conclusion of this sequence, the space on the source volume formerly occupied by the catalog is now owned by the copied version (the target catalog) and is available for suballocation. Note that immediately after step #2, there are technically two catalogs that own the same volume - the source catalog owns itself and the target catalog owns the volume upon which the source exists. If a user tried to reference a cluster immediately after step#2 through the source catalog, or through the new target catalog, VSAM would permit both! This is a violation of VSAM volume integrity. You are supposed to guarantee that this violation does not remain in effect, i.e., you are supposed to insure that steps #3 and #4 are successfully executed. If you fail to complete the procedure, or allow usage of either the source or target catalogs before the procedure is complete, unpredictable results can occur. In a practical environment, this means you need exclusive control of the catalogs until the procedure is complete and successful.

Final comments on the nonrecoverable UCAT.

There are always going to be special situations that are not described in any publication. In dealing with VSAM and VSAM problems, I invariably hear: "I tried what you said and it didn't work." After much investigation of such claims, I find few that are directly attributable to VSAM. Here is a list of the more common faults that brought about the statement that VSAM 'didn't work'.

1. Someone did restore the catalog using IEHDASDR after all!
2. Someone restored a volume owned by the catalog.
3. Someone used the wrong coding in the CAT subparameter to force catalog operations to an incorrect catalog.
4. Someone used DELETE with the FORCE option incorrectly.
5. Someone used SUPERZAP instead of ALTER REMOVEVOLUMES to 'clean up' a volume.
6. Someone tried to SUPERZAP a part of the catalog.

In all cases, this 'someone' is not VSAM trained. In many cases, this 'someone' had the approval of a manager who is not VSAM trained. Further, in almost all of the cases, the 'someones' and their managers feel that the VSAM catalog is just like the old SYSCTLG - it's just that 'we don't know the new VSAM command language yet'. These people are prone to experiment with VSAM commands and subparameters 'until it works'. After experiencing such things, I feel you should take the following steps:

A. Train all people who will manipulate the catalog throughly.
B. Alert all managers to the dangers of experimenting with a catalog.
C. Set up a 'worst case' recovery procedure for every UCAT to discourage experimentation.
D. Make sure operations understands the ramifications of restore, whether the volume has a catalog on it or not.
E. Make sure operations understands that canceling a VSAM operation like a DEFINE or DELETE can lead to later catalog errors. This is especially true in MVS which permits the FORCE option of the CANCEL command.
F. Set up a procedure for handling VSAM catalog complaints at your own organization. In some accounts, I have found people afraid to complain about possible catalog errors, leading to the continued use of a defective catalog.

Generalized Recovery Procedure.

Most experienced users agree that about the only sure method for recovering a nonrecoverable catalog is to rebuild it. The basic reason is that the time factor involved in attempting some kind of SUPERZAP is prohibitive. If the old catalog is at least readable, you can do the following:

1. LISTCAT the old catalog.
2. LICTCAT the UCATs in the master catalog to find out the alias entries for the catalog to be rebuilt.
3. EXPORT each of the VSAM entries in the catalog to be rebuilt.
4. DELETE the UCAT. (This will also delete all the VSAM space associated with the catalog. You can use the FORCE keyword to delete the space even if it's not empty although I don't recommend it.) All aliases in the master catalog are removed with this command.
5. DEFINE a new UCAT.
6. DEFINE all required aliases in the master catalog.
7. IMPORT all of the cluster EXPORTed in step 3.
8. If nonVSAM entries were in the old catalog, redefine them using the listing from step 1 as your source.

RECOVERABLE UCATS.

When using a recoverable catalog, VSAM automatically generates a second copy of the catalog entry in an area called the CRA - the Catalog Recovery Area. VSAM will maintain both copies of the entry allowing you to use the second copy to recover a damaged catalog through the use of four special commands - LISTCRA, EXPORTRA, IMPORTRA, and RESETCAT. The idea is straightforward, but you should understand some things about CRAs at the outset.

First, a recoverable catalog is specified by coding the RECOVERABLE parameter in the DEFINE UCAT. Note that the MCAT need not be recoverable to define a recoverable UCAT. A CRA will be allocated by VSAM on the volume containing the UCAT and on every volume the UCAT subsequently owns. The CRA is initially allocated 1 cylinder with the space suballocated from normal VSAM space unless you use either the UNIQUE or CANDIDATE options in defining a cluster or volume respectively, in which case DADSM is used to obtain an extra cylinder. This means that you must allow for the CRA when using a recoverable catalog. If you define a small UCAT, the CRA space is suballocated from the space given to the UCAT and may be a significant part of the space you thought was allocated to the catalog. If the UCAT definition is extremely small, the DEFINE can actually fail. Note that this CRA space is not shown in the catalog space calculations described in the AMS SRL; it is assumed you know enough to add the extra cylinder to your calculations. Secondary allocation is handled in the same manner as the primary allocation up to a maximum of 15 cylinders for a total maximum of 16 cylinders.

Second, you should be aware that there is only one copy of the catalog entry in one CRA. In the case of multi-volume allocations, the space map in the CRA on all volumes other than the first will account for the space allocated to the multi-volume object, but there will not be copies of the catalog record in all of the CRAs. The implication is that you need to know which CRA holds the duplicate entry in order to use the commands that effect catalog recovery. The basic rules are:

KSDS	First volume allocated to the index component.
ESDS	First volume allocated to the data component.
RRDS	First volume allocated to the data component.
AIX	Use the same volume used by the base cluster.
PATH	Use the same volume used by object over which the path is defined.
UCAT	Use the volume containing the MCAT. (This is the UCAT entry, not the UCAT's self-defining records.)
NONVSAM	Always in the CRA on the volume upon which the catalog in which they are defined resides.

Third, from the above paragraph, you should realize that there is a limitation as to how many nonVSAM entries you can put in a large UCAT. If the UCAT is 100 cylinders, you cannot use all 100 for nonVSAM entries because those nonVSAM entries must be duplicated in the CRA on the catalog's volume, which is a maximum of 16 cylinders for all purposes. If you need extensive nonVSAM entries, you should either use multiple UCATs, nonrecoverable UCATs, or CVOLs. The choice is really up to the installation. My own first choice would be UCAT because that will simplify the recovery procedures due to the fact that all catalogs are of a single type - VSAM. However, VSAM catalogs do take more space than CVOLs, and installation that already have CVOLs will probably stick with them because they are well understood at such installations. The net is, you will have to make up your own mind on this one. The performance of VSAM nonrecoverable UCATs is very close to that of a CVOL, so the real issue is one of standards rather than performance.

Fourth, it goes without saying that commands NOT designed to manipulate the CRA do not process the CRA in any special way. Thus, the REPRO command will not process a CRA which is why, in MVS, REPRO will not copy a recoverable catalog, and why, in all systems, the user is expected to validate a catalog reloaded from an unloaded version of the catalog via REPRO.

Finally, remember that this section deals with UCATs, not the MCAT. A UCAT should not have any UCAT or CVOL entries in it since that function is reserved for the MCAT only. However, it is possible to place such entries in a UCAT so that the UCAT can later be used as an MCAT. This function is used commonly during system generation. Notice that it is presumed that if you put such entries in a UCAT, you will use the UCAT as an MCAT at some future time; thus there is no easy way to delete such entries from a UCAT while it is still a UCAT! (The CATALOG parameter of DELETE cannot be used to direct DELETE to remove a UCAT entry from a UCAT.) One way of handling such errors would be to use the RESETCAT command to reset into a newly defined UCAT of the same name on a different volume. This type of processing will be described later.

SUPERZAP and RECOVERABLE UCATS.

The purpose of recoverable catalogs is to allow you to use VSAM recovery tools to make catalog recovery easier. It should be obvious to you that if you have a recoverable catalog, you cannot simply SUPERZAP an entry in the catalog to 'fix' something because there is a duplicate of that 'something' in the CRA. Of course you could print the CRA and SUPERZAP it also, but by the time you did all that, any shop with a well organized recover procedure could have beat you ten times over. There are always special cases, but in this section, I am definitely assuming that the cost of SUPERZAPPING is simply too high in 99% of the cases, so I am never going to recommend that you use it. Now remember, this does not mean you should not learn how to use SUPERZAP, or that you should decide to not learn how a VSAM catalog works, etc., etc. There will always be that 1% that defies reasonable recovery procedures. It's just that I am not writing about that 1%.

Detection - how do I know I have a bad recoverable catalog (UCAT)?

Detection of a problem with a recoverable UCAT is similiar to that of detection of a bad nonrecoverable catalog - many data sets cannot be opened, VSAM cannot open the catalog itself (IEC161I), or VSAM encountered I/O errors in the catalog (IEC331I, IEC332I, and IEC333I). As stated previously, if you can at least open the catalog itself, you can begin the recovery, probably by using LISTCRA with the COMPARE option to establish the extent of the damage. But that's getting too far ahead. Let's concentrate on the case where the catalog itself cannot be opened first, getting into recovery of a partially damaged catalog later.

If you cannot OPEN a catalog, several things could be at fault:

- Not enough storage was available for VSAM to build the necessary control blocks, load the proper routines, etc.
- No unit was available to satisfy a mount.
- A user supplied catalog name does not match the type U record in the MCAT and the user supplied catalog name is correct.
- An I/O error occurred while attempting to read the catalog's cluster record, the FMT1 or FMT4 DSCB on the catalog's volume, or the JFCB could not be read.
- The FMT1 DSCB or the catalog's cluster record is invalid.

The first two situations can easily be corrected through JCL. A bad type U record in MCAT is a bit more difficult because you cannot just go ahead and DELETE a UCAT from the MCAT when that UCAT does not exist. This is because VSAM will try to mount the catalog's volume in an effort to delete the DSCB in which the catalog exists. Further, VSAM will release volume ownership in a successful deletion of a UCAT, but in this case there is no volume to access!

If you decide to just leave the bad entry in the MCAT until the next time you clean up MCAT, and then you try to simply DEFINE the already existing UCAT in MCAT, you will get a volume ownership conflict. This situation is similiar to the UCAT entry defined in a UCAT described in the first paragraph on this page, and the fix is the same - in this case recovery of the MCAT.

If I/O errors occur in processing the FMT4, it is equivalent to total destruction of the volume. Presumably, the catalog, CRA, and nonVSAM data sets cannot be located because the VTOC is in error, so you will definitely need to restore your backup and repair as required in this case.
If the I/O errors occur only in VSAM entities, i.e., the cluster catalog record, or the FMT1 DSCB for the space occupied by the catalog, (or the records mentioned are invalid), then recovery must proceed for the whole volume with respect to VSAM. Since the nonVSAM data sets are theoretically processable, you can technically restore only the VSAM spaces on the volume (using the partial restore feature of IEHDASDR) and begin the catalog recovery process. Note that if the CRA itself is not damaged (it's pointer is in the FMT4), you can effect recovery using it. You must carefully evaluate partial restores however, because the added complexity in trying to maintain exact track locations may offset any added value.
My point here is simple - you still need to understand and use backup procedures with recoverable catalogs. Recoverable catalogs will lessen the impact of a disaster, and make recovery much simpler, but they do not prevent disasters.

If the JFCB cannot be processed, the error is probably not in VSAM because VSAM does not build the JFCB. In systems with DASD job queues (VS1 and SVS), you can try rerunning the job because you will probably get the JFCB positioned at a different location. If that fails, or you are in MVS, consult the system programmer assigned to the SCP as you probably have an SCP failure.

RECOVERY OF THE RECOVERABLE UCAT.

I am going to first show you the procedure for recovering a recoverable UCAT when the UCAT cannot be opened. The procedure is a generalization of several techniques and is NOT meant to fit every case. Rather, it's intent is to describe some of the vagaries of recovery of a recoverable catalog. The procedure assumes a complex environment with multivolume clusters existing partially on the volume to be recovered. My hope is that after reading about this procedure, you will understand what critical points MUST be taken into account in planning a recovery procedure. You may decide to junk the entire procedure in favor of IEHDASDR and rerunning jobs. As long as it works for you, and you understand it's restrictions, fine - go ahead and use it. I just want you to know how to deal with a recoverable catalog. The procedure consists of 16 major steps:

1.	LISTCAT ENT(catname)ALL	Get all the UCAT aliases from MCAT. (MVS only.)
2.	EXPORT(catname)DISCONNECT	Prevent cat. from being used, but deletes aliases in MVS.
3.	EXPORTRA ALL on catalog vol.	Unload all VSAM clusters and nonVSAM entries via CRA.
4.	EXPORTRA ENTRIES multivolume	For multivolume clusters that reside partially on the catalog volume.
5.	ALTER REMOVEVOLUMES	Clean up the catalog volume.
6.	IEHATLAS	Does alternate track assignment for bad tracks.
7.	DEFINE UCAT plus all aliases removed in step 2.	Remember that primary space must be sufficient to hold backup copy of the catalog.
8.	REPRO	This is a reload of a backup (unloaded version) of catalog.
9.	DELETE multivolume clusters described in step 4.	Prevents getting duplicate name error for UNIQUE clusters to be imported via IMPORTRA later.
10.	IMASPZAP FMT4 timestamp to agree with catalog T/S.	If you don't, LISTCRA and RESETCAT in following steps can't open the catalog.
11.	LISTCRA COMPARE	For all volumes owned by the catalog.
12.	IEHDASDR	Backup all miscompared and multivolume clusters in event subsequent RESETCAT fails.
13.	RESETCAT the miscompares.	Don't forget the multivolume clusters in command syntax.
14.	DEFINE space as required.	
15.	IMPORTRA	Clusters and nonVSAM entries from step 3.
16.	IMPORTRA	Multivolume clusters.

Discussion.

It should be immediately obvious that the procedure assumes multivolume clusters, a damaged track, and an unloaded backup of the catalog via REPRO. If you have no multivolume clusters, steps 4, 9, and 16 can be eliminated and if you do not have any bad tracks you can eliminate step 6. So while the length of the procedure is at first frightening, in many cases you will not need all of the steps. I said that earlier.

The LISTCAT (and all that follows) presumes you can at least open the catalog. The non-openable catalog was covered earlier. The LISTCAT is required because the subsequent EXPORT DISCONNECT will delete aliases from the MVS MCAT which you have to put back later with the DEFINE in step 7. If you are not in MVS, you could eliminate this first step, but in any event, it's nice to have a record of what you started with in the recovery.

The EXPORT DISCONNECT prevents the catalog from being used while you are in the process of recovery. Further, you may decide to move the catalog to a different volume (step 8 would be omitted in this event) and in any case, the redefinition of the catalog in step 7 will require the MCAT to be free of any U type entry that names the UCAT.

The EXPORTRA ALL for the catalog volume generates a backup copy of eveything on the catalog volume. This EXPORTRA is set up for only the CRA on the catalog volume so multivolume clusters are not exported as noted in PART I PAGE 14. It is possible to define a single EXPORTRA to do the whole job, but practical experience has found that the multiple EXPORTRA commands are easier to understand and handle. (Quite a few tapes may have to be mounted to contain a multivolume cluster and most customers like having the very large clusters placed on separate sets of tapes.) Note carefully that this EXPORTRA will export all the nonVSAM entries defined in the catalog. It's probably a good idea to go back to PART I PAGE 14 and reread the EXPORTRA information over again at this point.

The EXPORTRA for the multivolume clusters is put in the procedure for the case of a multivolume cluster that resides partially on the catalog's volume. If the multivolume cluster lies wholly outside of the catalog's volume, the later RESETCAT should handle that entry. You can see now that the physical location of multivolume clusters can affect recovery. If there are many multivolume clusters that intersect in multiple ways across multiple volumes, recovery can be quite complex; if you can all multivolume clusters on volumes separate from the catalog volume, with no intersections, then recovery can be simplified.

The ALTER REMOVEVOLUMES is used to clean up all traces of VSAM on the catalog volume without disturbing the catalog's owned volumes. If DELETE were attempted here, it would fail because the catalog is not empty. This leaves the CRAs on the owned volumes intact for the upcoming RESETCAT.

The IEHATLAS step is there simply to indicate that this is the time to repair (or rather reassign) bad tracks. You could of course use IEHDASDR and initialize the whole pack if you like, but the procedure is meant to show what you could do in the event of a partial pack failure, i.e., the nonVSAM data sets are still on our defective pack; the only bad track is one within the catalog. This step, and the one that follows, will be the subject of much discussion, because I'm sure that many of you will say: "Why don't you just restore a backup copy of the catalog using IEHDASDR and start from there?" In many cases, probably the majority of cases, I would have to agree, especially if whole VSAM volumes are predicated. But, and this is a big BUT, recall that the CRA is pointed to by a field in the FMT4 DSCB, not the catalog. If you attempt partial restores, you had better make sure that the CRA ends up on the same tracks as it dis before. All of this can be done. The point is, did you do it?

Because I want to show you how to use REPRO in the backup of a catalog, and because the partial restore of a VSAM volume can lead to problems with partially restored CRAs etc., and because I am assuming that my volume contains some nonVSAM data sets that I am loathe to restore, I will use the REPRO as one of my two prime recovery vehicles. (RESETCAT is the other one.)

The REPRO function does not do a DEFINE, so that means you have to do the DEFINE before you can execute REPRO. Recall that when you do this define, you are going to reload an unloaded version of the catalog, i.e., this is not a copy function. (Copy is permitted only for nonrecoverable catalogs.) PART I PAGE 12 of this paper documents the unload/reload restrictions. Basically, you must reload into a catalog with identical characteristics on the same volume. Further, the reload will NOT reload the CRA on the target volume, so you 'fix' that using the other steps in this procedure. You can postpone the definition of all the alias entries as long as you like; I show them here to remind you that you have to do them.

REPRO processes the catalog just like it processes any other data set, i.e., it merges records into it. At first this seems quite simple and without complication, but consider the timestamps. There are three timestamps to consider:

FMT4 DSCB	Now at time T1.
Volume (Catalog)	Now at time T1.
Volume (CRA)	Now at time T1.

All three timestamps are at the same level so REPRO can open the catalog as a data set and process it without encountering timestamp mismatches. Now REPRO never modifies the CRA in this process, so due to the fact that REPRO is reloading a back level version of the catalog, the timestamps immediately after the reload could appear as follows:

FMT4 DSCB	Now at time T1. REPRO did not change this one.
Volume (Catalog)	Now at time T0. REPRO modified this one due to catalog merge.
Volume (CRA)	Now at time T1. REPRO never modifies the content of the CRA.

Command like LISTCRA and RESETCAT open the catalog as a data set, not as a catalog. As such, normal timestamp checking prevails. The open will detect the timestamp mismatch between the FMT4 DSCB and the volume record in the catalog and will fail. To get around this, you must make the FMT4 DSCB timestamp equal to that of the volume record in the catalog. The only way to do this is to SUPERZAP the FMT4 DSCB to agree with the timestamp in the volume record. The timestamps will now read:

FMT4 DSCB	T0.	Due to SUPERZAP.
Volume (Catalog)	T0.	
Volume (CRA)	T1.	

As explained in PART I PAGE 3 and following, the mismatch of the CRA timestamp will NOT prevent VSAM from opening the catalog and using it, although the CRA will not be updated unless it agrees with the volume timestamp in the catalog. Fortunately, the RESETCAT command of step 13 will force all timestamps back to T1. Presumably, you are not going to attempt to use the catalog until you complete this procedure, so the RESETCAT can wait until after you do a few more things to this partially restored catalog.

While on the subject of timestamps, I should point out that this problem does not exist on volumes that do not contain a catalog. If you restore a non-catalog volume, it's timestamps will probably be out of synchronization with the catalog, but a RESETCAT of the CRA into the owning catalog will fix that.

Notice here that the DELETE of the multivolume data sets occurs after the REPRO but before the SUPERZAP. After the discussion of the timestamp problem, this might appear impossible, but the point is that DELETE does not open the catalog as a data set - it opens it as a catalog. Thus some VSAM commands might work immediately after the REPRO while others would fail. In practice, you could interchange steps 9 and10 with no effects.

The LISTCRA COMPARE in step 11 is obvious - you want to find out just how bad the restored catalog is. If the reload is from a recent unload, you might have very few miscompares. Only you can judge the frequency of unload necessary to minimize the miscompares. Since REPRO does not modify the CRA, it is probable that you will get miscompares on the catalog volume itself since it was completely rebuilt, but even that can be minimized by placing clusters on the catalog volume that change infrequently.

You might think the next step should be the RESETCAT. As you see, it is not - it's an IEHDASDR. The reason IEHDASDR is strongly recommended is that during RESETCAT processing, the catalog is always updated, but there are times when the CRA is also updated. If a failure were to occur during the time RESETCAT was processing an entry that required updates to both the catalog and the CRA (failures here include such things as power failures or some operator error like IPLing without bringing the system down properly), it is possible for partial updates to have taken place to both the catalog and the CRA at the time of the failure. These failures could render both the catalog volume and the volume with the CRA being reset useless for further processing with respect to VSAM.

The next question is why do all this IEHDASD processing for the multivolume clusters also - why not just the miscompares? Suppose there are three volumes involved in the RESETCAT processing - CATVOL, MV1 and MV2. A multivolume cluster resides on MV1 and MV2 but not on CATVOL. After the last unload of the catalog on CATVOL, the multivolume cluster was extended. The LISTCRA COMPARE will show that the multivolume cluster is not represented properly in the restored catalog.

Now let me make two assumptions about this situation:

1. The extension to the multivolume cluster occurred in MV2 not MV1.
2. You misread the LISTCRA COMPARE output and assume the error is on MV1 - or - you discover that the multivolume cluster is a KSDS with it's index on MV1. Since you remember that the CRA for a KSDS is on the first index volume, you decide to use only the CRA on MV1 for the RESETCAT

Now the assumption that most students make, especially in MVS which uses dynamic allocation, is that RESETCAT is some how going to 'find' all necessary CRAs and use them to RESET the catalog. If you specified NONE for a CRA involved in a multivolume cluster, you are telling RESETCAT not to use that CRA for RESET - use it only to perform consistency checks. RESETCAT will find that the index component of the KSDS is consistent but the data component is not. RESETCAT will then mark the data component unusable on both MV1 and MV2. The space assigned to the component may be either scratched or returned to the catalog for suballocation. Of course, by correctly coding the RESETCAT in the first place, the cluster would be properly reset, but the point I was making is that if you make a mistake in handling the multivolume cluster, it could be marked unusable. That is why the IEHDASDR dump is recommended for all volume involved in a multivolume reset.

A logical question at this point is: "Yes, but I thought we used EXPORTRA to save all the multivolume clusters in step 4. Why all this backup now?" Well, remember that the EXPORTRA in step 4 was for multivolume clusters that were known to be on the catalog volume. The IEHDASDR described here is for multivolume clusters that are found to be miscompared after the LISTCRA COMPARE of step 11, that is, multivolume clusters that are contained on volumes separate from the catalog volume.

The next step, the RESETCAT itself, is key to understanding the entire procedure, so I will go into it in some detail. You should read the RESETCAT information in PART I PAGE 16 of this paper and also the description of RESETCAT processing in the AMS manual before going on.

RESETCAT PROCESSING.

The basic processing of RESETCAT is to use a workfile, which is an RRDS cluster, into which the LKR control intervals of the catalog are copied. The CRA records are then merged into the workfile. The workfile is then used to update the catalog. Since the workfile is an RRDS cluster, it must be cataloged in a VSAM catalog; the catalog used for this purpose cannot be the catalog that is being reset - it is termed the WORKCAT in most literature.
From this, it is obvious that you must be able to allocate the WORKCAT and the RRDS cluster. The size of the RRDS cluster is normally no larger than the catalog to be reset, but during RESETCAT processing, the size of the catalog might increase due to the fact that the reloaded catalog is badly out of sync with the CRAs. The workfile must be able to handle the additional entries as well as the resultant catalog. If a catalog is extended, it is extended in increments of 6603 records where each record is 505 bytes plus the 7 bytes required for the RDF and CIDF fields making a total of 512 bytes for the CI size.
The actual formula for the secondary allocation is:

$$\text{Secondary} = \frac{(\text{MAXCI}*2 - \text{primary}) + 125}{126}$$

Primary = size of the current catalog.
MAXCI = largest CI number possible for a catalog.

The logic flow through this initialization stage is as follows:

- Open the catalog as a cluster. Check to make sure the catalog is recoverable, and is not the current master catalog. (The current MCAT cannot be reset.)

- RESETCAT guarantees that it has exclusive control of the catalog by issuing two macros - USYSINFO and URESERVE - which result in an exclusive ENQ using the name SYSIGGV2 and the catalog name.

- The workfile is defined. (It cannot be defined using the catalog to be reset.)

- The CRA volumes are now allocated depending on the coding in the RESETCAT command. If no CRA volumes are specified, an error occurs and RESETCAT terminates.

- The catalog's LKR is then copied to the workfile. All records that indicate that they 'belong' to a CRA that is going to be used to reset the catalog are flagged as deleted so as to later allow the merge operation to replace them with the copies in the CRA.

- The CRA records are now merged into the workfile. The three basic rules are:

Record exists only in CRA	Insert entry into catalog
Record exists only in the catalog	Delete entry from the catalog
Record exists in both the CRA & catalog	Replace catalog entry with entry from CRA

- The catalog record associations are checked for validity and a space consistency check is also performed. The space consistency check uses bit map techniques to zero out space occupied by a cluster. If, in trying to 'zero out' the bit map for a cluster it find the map already zero in the appropiate are, a consistency check will occur and the cluster will be marked unusable and any space that remains (i.e., if only part of the cluster overlapped) is made available. The association check validates that the type D and I records are associated with the correct type C record and that the DSDIR (data set directory) in the volume record matches also. Control Interval numbers are checked for record types C, B, A, U, and X and corrected if necessary. A special check is done for multivolume clusters. The amount of checking depends on whether all or only part of the volumes involved are specified as reset volumes. If all of

the volumes are specified as reset volumes, then RESETCAT can check all volumes and the updated catalog will reflect this checking, i.e., the cluster will either be validated or it will be deleted.
If only part of the required volume list was specified, then RESETCAT can validate only part of the multivolume cluster. The amount of checking depends on whether you specified the primary CRA (the CRA that contains the duplicate records about the multivolume cluster) as a reset CRA or not.

Primary Specified:

Replace the catalog entry with the CRA entry.

Primary NOT Specified:

Check space only on the specified volume.

Additional checks with respect to define time timestamps are done in all cases, as well as a DSDIR check even if the primary CRA is not specified; in this case the catalog records are used to validate the DSDIR instead of the CRA records.
If a check fails, the cluster is normally marked unusable and it's space freed up as described earlier.

- The catalog is now updated. It is during this process that RESETCAT may have to rename an entry it is about to insert. If RESETCAT finds that the name of the entry it is about to insert already exists in the HKR, it generates a different name of the form Txxxxxxx.VSAMDSET.DFDyyddd.Tyyyyyyy.Txxxxxxx, inserts it, and gives you a message which lists both names and which CRA was involved. Most students inquire at this point why the reason for the rename at all because they feel all of that was handled during the merge. A simple example serves well to illustrate the point:

  A nonVSAM entry A.B.C exists in the catalog.
  A volume is restored and RESETCAT run against it.
  RESETCAT discovers a VSAM cluster on the restored volume with the name A.B.C.
  RESETCAT cannot destroy the nonVSAM entry so it renames the VSAM cluster.

  Note that inserting such entries into the catalog requires room in the HKR for the additional names. This is one reason the catalog MUST BE EXTENDABLE during a RESETCAT.

- The CRA is updated if required. At least one example of where it's required is if a space consistency check fails and a cluster is marked unusable. Remember, in that situation, the space is returned to VSAM so the volume record in the CRA, at a minimum, would have to be updated.

- Final processing. Close all files, delete the workfile, write all messages, etc.

Key Points about RESETCAT Processing.

The CRA can be updated during RESETCAT processing. Because an external failure such as a power loss could interrupt the processing at a point where partial updates have been done to the CRA, you should back up all volumes taking part in the RESET.

Special checking is done on multivolume clusters. If one volume of a multivolume cluster fails it's check, the part of the multivolume cluster on that volume can be deleted and the cluster marked unusable in the catalog. The procedure I'm describing used EXPORTRA and DELETE to remove multivolume clusters that would have participated in the reset. You must make a decision as to whether or not that kind of backup is worth it in your environment. If you normally EXPORT the multivolume clusters anyway, you might

decide to eliminate the multivolume EXPORTRA and see if the RESETCAT is successful. If not, you can delete the cluster and IMPORT your backup copy.

It seems obvious to me that if you restore only one volume of a multivolume cluster and attempt to use RESETCAT, errors are quite likely. It is up to you to guarantee that all volumes of a multivolume cluster are at the same level.
Further, it is of course possible to restore a single volume of a multivolume cluster in which the data has been modified, but the extents are identical. VSAM has no way of knowing this, if the volume timestamp also happens to be the same. Since you might have a KSDS with the index component on a different volume than the one restored, unpredicable results can occur when you attempt to process this data.

As described above, the RESTCAT can rename certain entries. If the entry is a nonVSAM entry, it's corresponding FMT1 DSCB is NOT renamed because the nonVSAM data set can be cataloged in multiple catalogs. You must take some action if you want to use the entry RESETCAT renamed. Note that if you decide to change the name in the FMT1 to agree with the name assigned by RESETCAT, a data set not found condition will result if you try to access the data set through any of the other catalogs or directly through JCL using the old name.
If the renamed entry is a UNIQUE cluster, RESETCAT will update the FMT1 DSCB for you. Since your JCL probably has the old name in it, again, you must decide what to do about the two like names.

It is possibe for RESETCAT to rename a GDG entry in the MVS system. Unfortunately, as of this writing, the ALTER command does not permit renaming a GDG entry. The only solution for this situation would be SUPERZAP. I'm tempted to state here that any account that allows duplicate GDG base records to exist anywhere in the system deserves SUPERZAP, but I suppose you could build a case for it if you acquired a company which happened to use the same GDG name as yours.

I want to make one important point, if I make no other in this section. You simply cannot allow untrained people to reset a catalog. Even if you have a 'foolproof' procedure, who is going to read the messages from RESETCAT and understand what to do to correct any problems? I have met several managers who refuse to allow people to be trained in VSAM claiming it is not necessary. If you are one of those people, I predict you will lose your catalog at some point, and you will be so mixed up that you will have to start over from scratch to rebuild it. It is beyond my comprehension why anyone would disallow VSAM training for at least the system programmers responsible for the areas involved.

Steps 14, 15, and 16 simply redefine required space that may have been lost and bring back the clusters exported via EXPORTRA in steps 3 and 4. I don't think I need to go into detail about these. This ends the discussion about the normal operation of the procedure that was described on page 8 in this section. Remember, this discussion assumed the catalog was not able to be opened. It also assumed there were no I/O errors while RESETCAT was processing. In the next paragraphs, I will discuss these points.

RESETCAT HANDLING OF I/O ERRORS.

In the procedure just described, it was assumed that RESETCAT executed successfully. I/O errors can occur in either the catalog or the CRA, so you must take this into account when you design your procedure. A student might ask: "Why are you considering I/O errors? Was not the catalog restored in the previous procedure? If it was restored, wouldn't you get the I/O errors on the restore?" The answer is, yes, you will get I/O errors on the restore. But RESETCAT is not ALWAYS used to reset a newly restored catalog. Further, I/O errors are possible in the CRA whether you restore the catalog volume or not, if the CRA volume was not also restored. So I/O errors are a possibility in every case.
At the outset, you should understand that RESETCAT does not do dynamic alternate track assignment. That's why step 6 was in the procedure in the first place - it warns you of this fact. Two cases are possible:

- a track is defective and no alternate has been assigned.
- an alternate has been assigned to a defective track, but the data could not be recovered.

RESETCAT treats both of these cases alike - both appear as a defective track. There is a limited capability of handling I/O errors through the IGNORE option of RESETCAT. More about that later.

Suppose a catalog I/O error occurs.

Since RESETCAT does not check the HKR (except to insert names into it), a defective track in the HKR will go undetected.

If NOIGNORE is specified for RESETCAT, a permanent error in the LKR will cause RESETCAT to terminate immediately. If IGNORE is specified, the records on the defective track in the LKR cannot be copied to the workfile, but the corresponding records from the CRA can be treated as a 'new' cluster and inserted into the catalog, effectively repairing the defective track.
During the insert, RESETCAT will encounter the same name in the HKR, i.e., the names of the objects that occupied the defective track. Because these objects are being treated as 'new' objects by RESETCAT, renaming will take place as described earlier. The true names will still point to the defective track and the entry will have a generated name in the HKR pointing to the 'new' control intervals assigned by RESETCAT for this 'new' object. It is up to you to delete the true name and ALTER the generated name (if possible - remember GDG) after RESETCAT. Messages are issued as described in the RESETCAT discussion after the procedure, and the defective tracks are marked unavailable in the catalog.

Suppose a CRA I/O error occurs.

If NOIGNORE is coded, RESETCAT will terminate immediately and no resetting of the catalog will occur. If IGNORE is coded, the defective track in the CRA is simply bypassed. Since records read from the catalog into the workfile are flagged as 'deleted', this situation will be reflected in the catalog as a deleted object. Their true name entries will also be removed from the HKR, but unlike the catalog I/O error case, the defective tracks in the CRA will NOT be marked unavailable in the CRA's CCR (Catalog Control Record).

The net effect of I/O errors is that they cause RESETCAT to terminate immediately, with no resetting of the catalog, if NOIGNORE is chosen. If IGNORE is coded, RESETCAT will recover as much as it can, but inconsistent objects will be deleted and their space made available for reassignment. Now messages to this effect are issued, but the point is: "DO YOU HAVE BACKUP FOR THOSE DELETED OBJECTS?"

Suppose there are permanent CRA errors.

Since RESETCAT treats the CRA as a data set, it must be able to be opened. If the CRA's self defining records are damaged, the CRA is not recognizable as a CRA and RESETCAT processing will immediately terminate with no action. If the CRA's volume record is damaged, RESETCAT will continue only if IGNORE is specified, but then all the clusters on the volume will be marked 'not usable'. EXPORTRA (and other commands that open the CRA) is also not able to function with these kinds of CRA errors. You are made aware of these errors via messages IEC331I, IEC332I, and IEC333I with a return code of 028-008. The 008 indicates CRA problems instead of catalog problems.

Empirical testing has shown that CLOSE will fail with a feedback code of 180 (x'B4') if you try to close a cluster that has bad entries in the CRA. This is due to the fact that the cluster's CRA records must be updated with statistical information when CLOSE or CLOSE TYPE=T is executed. A program that tests CLOSE feedback codes can indicate the error at an early time, but the question is, if such an error is indicated, what can be done? Obviously, if you have a CLOSE error, you may not be able to recover this cluster at a later time, or you can misinterpret a LISTCRA printout and assume the cluster is bad when in fact you can get at it through the catalog.

Happily, you can use the EXPORT command in this situation. You will get message IDC3351I with a CLOSE error return code of 180 and a condition code of 12, but the portable data set will have been correctly created and can be processed by subsequent IMPORT commands. To find out how many clusters are involved in the damaged area of the CRA, use LISTCAT (volser) SPACE ALL to determine which clusters are on the volume in question. You can then EXPORT all of the clusters printed. You could try to limit the number of EXPORTs to just the clusters you feel are in the damaged area by using LISTCRA COMPARE to determine if any clusters compare successfully. However, since it is wise to repair the CRA rather than 'let it go for while', you probably should EXPORT all the clusters, DELETE the space on the affected volume, reDEFINE the space, and IMPORT the clusters. ALTER RVOL may have to be used as well together with DELETE SPACE NOSCRATCH.

REMEMBER, THE AVAILABILITY OF RESETCAT DOES NOT ELIMINATE THE NEED FOR OTHER RECOVERY TOOLS!

A Pause to Look Back.

At this point, I'll bet you are wondering whether RESETCAT is worth it. After all, it seems you still need to devise recovery techniques along the 'old' lines when RESETCAT fails, and there is a relatively slight performance degradation in order to use CRAs in the first place.
But keep in mind that while recovery of a catalog with few entries is relatively simple and fast, as that catalog grows, the time element alone might negate your present procedures. With nonrecoverable catalogs, you must ALWAYS start from 'scratch' to rebuild them; with recoverable catalogs, you at least have the option of using the new recovery tools. At this point in time, almost all users who have tried the 'quick fix' of either type of catalog through SUPERZAP agree that in most cases it doesn't work out. It almost always appears to have worked at first, but later it is shown that something was forgotten in the rush, and a much more difficult and time consuming recovery is required.

Second, realize please that the discussion about recoverable catalogs in this handout goes through some very fine technical points. After all, what are your chances that a track in the CRA will go bad? It can happen of course, and planning for it has that maddening characteristic that it never happens if you do plan for it, and it does if you don't, but what about the times when a user swears that either the catalog or the CRA is in error and you use a simple LISTCRA to prove the cluster is O.K.?

Then there is the performance argument. Obviously, any VSAM function that modifies the catalog in any way will have to modify the CRA also, and that will take so many extra EXCPs. It is just as obvious that IBM is not going to force you to use recoverable catalogs, just as we don't force you to use SMF. But, just as early users of OS found that SMF was worth it even though the overhead turned out to be about 5%, I think you will find that the cost of maintaining the CRA is worth it IF YOU KNOW HOW TO USE THE RECOVERY TOOLS DESCRIBED.

Then there is the question of building the CRA for the catalog's volume on that same volume. Would it not be far better to build it elsewhere, so that if the catalog's volume was physically destroyed, you could use the CRA to 'recover' the catalog's volume?
For the VSAM user, the only situation that might be improved if this was the case is if a track in the catalog was physically destroyed; if the whole volume is destroyed, all of the VSAM clusters on that volume are also destroyed, so of what value would the CRA be if it was somewhere else? But, in this situation, the CRA on the catalog's volume CAN be used to recover the catalog! So this particular argument is groundless.
For a nonVSAM user, the argument has some merit, because if the catalog's volume is totally destroyed, the nonVSAM entries are also totally destroyed, regardless of the location of the nonVSAM data sets themselves. I suppose I could philosophize here about the fact that you did not have that capability in the old SYSCTLG and VSAM was not designed to improve upon nonVSAM logic, but if you are an MVS user with 100% VSAM catalogs, the question will be raised.
The facts today are that there is no help in this area from recoverable catalogs. You will have to plan for this case in exactly the same way you plan for it when using nonrecoverable catalogs. That means, if disaster strikes, you had better be able to reconstruct the nonVSAM entries in any catalog! As I previously pointed out, why not execute LISTCAT from time to time, with the output on DASD or tape. If a volume containing the catalog is totally destroyed, use the information on the DASD or tape to construct DEFINE NONVSAM commands after the catalog is rebuilt.

Finally, realize that not every catalog must be defined the same. Those users who feel that for one reason or another, some catalogs should be nonrecoverable and the rest recoverable, can easily accomplish their goals.

THE MASTER CATALOG - VS1 AND SVS SYSTEMS.

These systems do not require the MCAT to IPL. Therefor, you cal IPL the system without VSAM, restore MCAT, then reIPL. Also, since MCAT is located by the system through the AMASTCAT CVOL entry in SYSCTLG, you can also change the AMASTCAT entry in SYSCTLG (through IEHPROGM) to an alternate MCAT and reIPL. The real question here is not how to do it, but how to 'guarantee' that when you do it, the alternate you select is valid, i.e., contains the proper entries.

To this end, it is recommended that MCAT be nonrecoverable and contain as little as possible. Effectively, this says MCAT should contain only pointers to UCATs, VSAM clusters that must be on the same volume as MCAT (due to volume ownership rules), and nonVSAM entries that you place in MCAT to take advantage of the fact that in these systems, MCAT will be searched first in an effort to locate an entry if VSAM is active.
If the catalog is nonrecoverable, you can use the REPRO command to unload/reload it. (See PART I for a description of the difference between 'copy' and 'unload'. These systems do not permit 'copycat' as of the time of this writing - January, 1979.) You can also use utilities like IEHDASDR to provide a backup.

Now you can make MCAT recoverable. You may be tempted to do this because you might want a user convention that states: "All VSAM catalogs .....". But before you disallow any exceptions to your convention, you might want to read about the MVS master catalog recovery in PART V. It is more critical, so if you follow the MVS rules, a later conversion to MVS will be much easier.

In setting up the actual recovery process, the first thing to check is whether you have all the UCAT records in the restored MCAT. Without them, jobstreams that use the UCATs will fail because the UCATs will be unable to be allocated in response to JOBCAT/STEPCAT statements.
Next, examine the recovered MCAT for inclusion of all nonVSAM entries that were there. If you have a recent LISTCAT of the old MCAT, you can use it to compare with the recovered MCAT. If that listing were on DASD or tape, you could write a program to simply extract the nonVSAM information, build a DEFINE NONVSAM command for each entry, and execute the command against the new MCAT. Entries already in the catalog will simple cause a duplicate name error.
It is obvious that you can't recover what you don't know about; users who put new entries on MCAT without reporting the fact to the control group can cause you lots of grief. It's their own making, but the VSAM group will still be castigated, so you might as well take the precaution of using an update level password to control who puts entries in the MCAT. When you define the password, you can also specify ATT(0), which means the operator will not be prompted for the password. That eliminates 'friendly' operators.

You might have some VSAM clusters defined in MCAT. This is not recommended because it further complicates the MCAT recovery. It is very easy for users of these systems, especially new users, to say: "We only have one VSAM cluster, so why do I need a UCAT?" If you have read all of this paper so far, the answer ought to be obvious; if it isn't, I'll restate it.
IF YOU MAINTAIN YOU WILL ONLY HAVE A FEW VSAM CLUSTERS TOTAL IN YOUR SYSTEM, WHAT PROCEDURE YOU YOU NOW HAVE FOR REEVALUATING THE RECOVERY OF MCAT IF THE NUMBER OF VSAM ENTRIES IN MCAT GETS ABOVE YOUR STATED LIMIT?

At any rate, if you have VSAM clusters in MCAT, you can recover them using IMPORT or REPRO. If MCAT was recoverable, you could also use EXPORTRA from the old catalog volume and IMPORTRA into the new MCAT. As stated above, it is not recommended that MCAT be recoverable however.

If the simple rule of making MCAT nonrecoverable and putting as little in it as possible is observed, then the users of these system can use IEHDASDR type programs to simply dump the MCAT volume periodically and restore it when needed. This is prpbably the choice of 99% of the users I know.
WARNING!
The MVS recovery is not as easy. Do not relax your rules because recovery of MCAT is 'easy' in your system.

Master Catalog Compatibility.

Within your own system, you should realize that your MCAT is identified in the VTOC of the volume by the high level qualifier of Z9999996 in the FMT1 DSCB name. UCATs have a high level qualifier of Z9999994. This does not prevent you from declaring (by changing AMASTCAT) that a UCAT is now the MCAT. VSAM will ignore the difference in the high level qualifier.

Because VSAM does ignore the difference in the high level qualifier of the FMT1 DSCB for catalogs, (it doesn't ignore the fact that the high level qualifier is Z9999992 which indicates no catalog in the space), a catalog can be defined in one system as a UCAT and made the MCAT in another system. Thus the user with two separate systems can recover the MCAT of one system on the other. It doesn't matter if the other system is DOS/VS, VS1, SVS, or MVS.

If you have an MVS system, or use an MVS system to recover your MCAT, you should be aware that the MVS master catalog has requirements that exceed the requirements of your MCAT. To handle these requirements, AMS has more command options. It is therefore possible to receive a recovered MCAT from the MVS system with page data sets and generation data groups defined on it. (This would be pretty poor recovery technique, but it's possible.) While you will be unable to delete these entries because AMS in your system does not recognize them, VSAM in your system will ignore the entries so you can use the catalog.
From the above, it is obvious that you cannot recover the MVS catalog on your system, except if the recovery vehicle is dump/restore logic.

If your system is down, but the MCAT is valid, you can IMPORT CONNECT the MCAT as a UCAT on another system and use it to process you data on that system. It is actually possible to 'share' an MCAT using this technique, but it is dangerous to do so because VSAM uses ENQ/DEQ logic as part of the safeguards for every catalog. Since ENQ/DEQ operates in only one machine at a time, while your MCAT is connected as a UCAT in another system it would be possible for the other user to delete some important entry such as SYS1.LINKLIB. For more information, see the section on catalog sharing.

THE MASTER CATALOG - MVS

MVS is quite different in it's MCAT requirements from other systems because in MVS, the master catalog is also the system's catalog. Because of that, MVS NIP processing searches the MCAT in an effort to locate required data sets. NIP locates the MCAT by locating the new entry SYSCATLG in the data set SYS1.NUCLEUS. You will recall that NIP can find SYS1.NUCLEUS because it must reside on the IPL volume. The SYSCATLG member contains the name and volume serial number of the volume that contains MCAT. There is no requirement for the older SYSCTLG structure at all. If used, it is simply termed the CVOL (control volume) structure.
This changes are responsible for a number of recovery considerations for the MVS catalog that are quite different from other systems. To understand their gravity, assume you have one CPU and MVS is down because of an MCAT problem.

- YOU CANNOT IPL THE CURRENT SYSTEM BECAUSE IPL REQUIRES MCAT.
  YOU CANNOT USE IEHPROGRAM TO CHANGE AMASTCAT BECAUSE THE IS NO AMASTCAT AND YOU CAN'T RUN IEHPROGM ANYWAY BECAUSE YOU CAN'T IPL.

- THERE ARE NO STAND ALONE VERSIONS OF VSAM UTILITIES. BACKUPS OF MCAT THAT RELY ON VSAM COMMANDS SUCH AS EXPORT AND REPRO CANNOT BE USED BECAUSE THE SYSTEM CANNOT BE IPLED.

- THERE ARE STAND ALONE DUMP/RESTORE UTILITIES SUCH AS IEHDASDR WICH CAN BE RUN. HOWEVER, THE SYSCATLG MEMBER OF SYS1.NUCLEUS FORCES MCAT TO A SPECIFIC VOLUME WITH A SPECIFIC NAME.

The above means that implementing an 'alternate' MCAT can be difficult. Note especially that if you use 3350 volumes, and the reason MCAT is bad is that the 3350 which contains it is bad, you must change the SYSCATLG member of SYS1.NUCLEUS before you can use a restored version of MCAT on another 3350, presuming that you don't want to change the serial number of a 3350. In effect, you can use an 'alternate' MCAT only if:

1. You are able to restore it in stand alone mode.
2. You are able to change the SYSCATLG member in stand alone mode, probably via IMASPZAP. (This assumes you know how to locate a PDS member in stand alone mode, or have previously recorded the exact DASD address of the SYSCATLG member.)

OR

3. Replace both 1. and 2. above with a small system which you will IPL to repair MCAT, operating on it as a UCAT to the small MVS system.

OR

4. Replace all of the above with modified NIP code to use some other member based on some switch to be set up by you.
   (THIS IS DEFINITELY NOT RECOMMENDED. THE PORTION OF NIP CODE CHANGED BY YOU IS NOT SUPPORTED BY IBM. FURTHER, IBM RESERVES THE RIGHT TO CHANGE INTERNAL CODE WITHOUT NOTICE. THIS COULD MEAN YOU SYSTEM MIGHT NOT IPL EVEN IF MCAT WAS WORKABLE, UNLESS YOU QUICKLY REEVALUATE YOU NIP MODIFICATIONS.)

Most customers agree that (3) is the best course of action. Use of (1) and (2) is O.K. if a restore is able to be done to the same volume as previously contained MCAT since then SYSCATLG need not be changed.

If you have two systems, the process is considerably easier, even if the second system is not an MVS system. In the first place, you can use the normal dump/restore and PDS utilities instead of stand alone versions if you intend to restore a backup copy of the catalog.
If the second system is an MVS system, it has all the capabilities of the first system, so you can proceed to check out all the entries in the newly restored MCAT for your other system. (It appears as a UCAT in the second system.) Specifically, you can manipulate page and swap data set entries, and can catalog GDGs and aliases. These types of operations are not permitted in nonMVS systems.
You can also use IMPORT, REPRO, and other VSAM commands because you are running in system state as opposed to stand alone.

If your second system is VS1 or SVS, you cannot explicitly DEFINE page or swap data sets, GDGs, or aliases, but you can run IMPORTRA, RESETCAT, etc. on the system. It is interesting to note that the RESETCAT command of VS1 will properly reset a MVS recoverable MCAT, i.e., it will handle page data set entries, swap data set entries, etc.
However, we shall see that it is not recommed to use RECOVERABLE as an option for MCAT in the first place.

Alias Entries.

As explained previously in this paper, the MVS system uses alias entries in MCAT to control the searching of both UCATs and CVOLs. Any restoration of any MCAT in MVS must take into account that these alias entries must be properly restored or else literally thousands of entries can end up on the wrong catalog. Since VSAM defaults to MCAT if it can't find an alias, those unwanted entries will appear in MCAT unless you have update level password protection. In that case, the entries will not be cataloged at all and confusion will reign supreme.

Remember that if your planned recovery procedure includes DELETE and/or EXPORT of UCATs, these commands also delete any aliases for the UCAT. You should take care to issue a LISTCAT before executing these commands:

LISTCAT CATALOG(mcat/password)ENTRIES(ucatname)ALL

At some point in the recovery process, you must guarantee that all UCATs and their aliases, and CVOLs and thier aliases are back in the MCAT. At least one customer does this by periodically executing the command shown above to tape. The tape is later used by a customer written program that builds the necessay DEFINE statements to restore the aliases.

Processing DISP=(,CATLG).

Let me also remind you that MVS must process requests from the scheduler to catalog nonVSAM entries in the proper catalog. To the user, this is recognized as handling the DISP=(,CATLG) parameter on the JCL statements. These requests are placed in:

- JOBCAT/STEPCAT is used.
- The catalog indicated by the alias in MCAT, which can be a UCAT or CVOL.
- MCAT itself, passwords permitting.

Thus, the MCAT in MVS also affects nonVSAM processing as well as VSAM processing. Unlike other systems, you get no choice in the matter here, so you have to do something about those alias entries. The basic recommendations for an MCAT in MVS follow.

Recommendations for a Master Catalog in MVS.

1. Keep it as small as possible.
2. LISTCAT your user catalogs periodically to maintain an accounting of aliases.
3. Make the MCAT nonrecoverable. This allows you to use the REPRO 'copycat' facility of MVS. Note that this facility of REPRO allows copying to dissimiliar devices, such as from a 3350 to a 3330.
4. Use at least an update level password and code ATT(0). This will prevent data sets that do not follow establised data set naming conventions from being cataloged inMCAT by default.
5. Plan the location of page and swap data sets carefully because the volumes upon which they reside will be owned by MCAT.

Let's go over these recommendations one by one. If you have just converted to MVS from DOS, the first recommendation might be difficult for you. Characteristicly, you have few, if any, UCATs and the rules for using them are probably nonexistent. This means you really need some sort of standards in your account now! Anything less is unthinkable.

Recommendation 2 is quite simple and should pose no problems. It might be nice, however, to start thinking about writing a program that builds AMS commands from the LISTCAT output. It's a great time saver. By the way, there is no need to get all involved in reading the catalog with assembler subroutines. VSAM uses a record format of VBA, logical record size of 125, and a blocksize of 629 for the output on SYSPRINT, so you can write a simple COBOL program, or any other kind of program to analyze the output.

The third recommendation might be more controversial, especially if you have some 'unbreakable' standard that says: "All VSAM catalogs will have exactly the same definition." Believe it or not, I have run into this, and it's tough to break because the rule was probably set up by someone who knows nothing about MVS and that someone is now in a position of power where any change to the standard would 'look bad'. About the only way out of that is to convince people that what you are doing is setting up a NEW standard for a NEW system; be sure to praise the old standard.
There is no denying that the availability of the REPRO 'copycat' facility is a major enhancement, and this might be your only reason for choosing nonrecoverable.
If you have a lot of VSAM entries in the MCAT, the performance of MCAT will of course be affected by having to update a CRA, but from recommendation one, this should not really be a factor.

About the only people who object to recommendation four are people who do not know how to handle passwords. Unfortunately, it is again the smaller user converting from DOS that has the most trouble with this. The usual case is a user who, in DOS, has been 'burned' using passwords and has now forbidden their use. A convincing case for passwords for at least the MCAT can be made by describing the power of the ALTER REMOVEVOLUMES command.

Recommendation 5 is the one that is probably the most difficult for you to implement. The difficulty involves the use of the REPRO copycat facility. If your backup procedure uses REPRO, it should recognize that REPRO effectively transfers volume ownership of the page and swap volumes to the alternate MCAT. Since it doesn't delete the entries from the now current MCAT (it can't - deletion of an open page or swap data set is prohibited), you now have effectively violated the rule of VSAM volume ownership - two catalogs now own the same volumes.
When you go to use your alternate MCAT, you might want to use the same page and swap data sets; your alternate may become your permanent MCAT in a sort of flip-flop arraingement. But if that is not what you intended, you will have to allocate the required page and swap data sets elsewhere and catalog them in the alternate MCAT. Since the naming convention of these data sets is fixed

by the name specified in the IEASYSxx lists in SYS1.PARMLIB, you may want to use a different convention for your alternate system. This has the advantage of keeping the page and swap data sets completely separate, which might actually be necesary, if you intend to use the alternate system to repair the MCAT and then attempt to warm start the main system. Obviously, if the trouble with the original MCAT is bad page or swap data sets, then you will have to use a second set in the alternate system, so most accounts take the position that it's better to be safe than sorry and use two sets of page and swap data sets. (Swap data sets are optional, so the alternate system need not use them.)

Before going any further, it is wise for you to reread the two sections in this paper on REPRO. They occur in PART I pages 11 and 12, and in PART III pages 2, 3, and 4. Further, there is a section in the AMS SRL entitled "Copy-Catalog Procedure". To find it, look in the table of contents under the major heading of "Copying and Printing".
Notice that the Copy-Catalog procedure has several steps, and that you may choose to stop the procedure immediately after the REPRO has been executed. This would leave you with two catalogs owning the same volumes, a definite violation of VSAM volume integrity.
Further, a special warning is given in the AMS manual to see to it that you quiesce the system before attempting to copy the master catalog. If you don't, it is possible for the MCAT to be updated in the midst of the copy. The validity of your copy is then in doubt.

On the other hand, if you choose the unload/reload facility of REPRO, there are possible changes you will have to make due to space conflicts. These conflicts are described in this paper on the pages mentioned, and in the AMS manual in the same area under the heading of "Reloading a Catalog". This particular problem should not occur for a master catalog, because if you follow the basic recommendations, the MCAT will be small and not contain many VSAM clusters in itself. Thus, the space maps in it should change infrequently.

Conclusions Up To This Point.

1. The VSAM master catalog is absolutely required to IPL MVS.
2. SYS1.STGINDEX, page and swap data sets are VSAM clusters that must be cataloged in MCAT. They cause MCAT to own the volumes upon which they reside due to VSAM volume ownership. Thus, these volumes are not available to effect any recovery operation that attempts to define a VSAM object on them through any catalog other than MCAT.
3. The MCAT is located through a member of SYS1.NUCLEUS called SYSCATLG. This member must be correct for IPL to occur.
4. The following data sets must also be cataloged on MCAT for an IPL to be successful:
   - All data sets with a first qualifier of SYS1.
   - All data sets that may be concatenated to SYS1.LINKLIB (through it's list in PARMLIB), regardless of their first qualifiers.
   - All data sets used in the start procedure for JES.
4. The MCAT may be copied or unloaded. In the case of MCAT, care must be taken to quiesce the system before this is done.
   The implication is that you might not be able to quiesce the system at the time you want to backup MCAT using REPRO logic.
5. Because of the above considerations, is is best to keep MCAT as small as practical, and to prohibit most users from cataloging in it via an update level password with ATT(0) specified.
6. If point #5 is observed, there is little need to use a recoverable catalog for MCAT. If you do use a recoverable MCAT, any use of the REPRO copy-catalog facility is prohibited by checks in the REPRO logic. The UNLOAD/RELOAD function of REPRO does not update the CRA! If you use it with a recoverable catalog, the catalog can't be used until CRA discrepencies have been resolved.

7. Most users elect to use a small alternate system to recover an MVS master catalog, assuming that a simple dump/restore type of backup is not sufficient.
   This alternate system will require all the data sets described in point # 2 and point #4 above.
   In particular, a second set of page data sets is required. Swap data sets are always optional for MVS.
8. It is not possible to operate on any open page or swap data set in the same system, because special flags in the UCB (as well as other VSAM control blocks) prevent it.
   WARNING.
   In a shared DASD system, it is possible to do this, if the MCAT of SYSTEM1 is a UCAT to SYSTEM2.
   This is to be considered a drastic error, and will definitely bring down SYSTEM1 eventually.
   It is therefor strongly recommended that you NOT share MVS master catalogs in this manner.
9. The actual names of page and swap data sets are controlled by parameters in the IEASYSxx lists in PARMLIB. Whatever conventions you decide to use, they must be consistent with these lists.
10. Alias entries in MCAT must be recovered if MVS is to use either UCATs or CVOLs correctly.

We now get into the exact recommendations to be used in the recovery of the MVS master catalog. I am going to pick a specific case, even though it might not fit your situation exactly. Remember, at the beginning I stated I can't hope to cover every case; rather my intention is to present at least one method of handling the problem, assuming the most common problems. You may then hopefully deduce what is best for you.

Type of Catalog Used.

Nonrecoverable catalog.
Update password protected, with ATT(0) specified.
CVOLs used, with aliases as required.
UCATs used, with aliases as required.
Minimum entries in MCAT at all times.
The catalog occupies all of it's space, to prevent VSAM clusters from being included in it's space.
The space does have secondary allocation specified, so as to prevent the MCAT from having to be reorganized at an inopportune time.

Types of Backup Utilized.

An alternate system will be maintained at the correct VSAM maintenance level so that a new MCAT can always be DEFINEd, even if all other backups are lost.

The dump/restore logic of utilities like IEHDASDR will be the first line backup. Because of this, MCAT will be located on a volume that has infrequent changes to nonVSAM data sets on that volume.

REPRO will be a second backup. It's primary purpose will be handle alias entries that may have been made since the last full backup. It is planned that REPRO will be run more frequently than the dump/restore because the REPRO will not affect nonVSAM data sets on MCAT's volume.

Because the cop-catalog facility requires the target catalog to be empty, the REPRO unload/reload function will be used instead. This eliminates the need to have a target catalog in the system, and also allows the use of tape as a secondary storage media.

At the same time the REPRO is executed, a LISTCAT of all UCATs will also be done to get a listing of the alias entries current at the time. If LISTCAT ALL is chosen, the MCAT is potentially rebuildable through redefinition alone.

Timing of the Backups.

From the VSAM point of view, the dump of the volume containing MCAT is only infrequently required; the real frequency is more likely determined by the volitility of the nonVSAM data sets.

The REPRO unload and LISTCAT are scheduled once per day at a time when the system can be quiesced or the online load is at least light. This minimizes the chance that MCAT could be updated just after a backup has been taken.

All users are informed of the backup cycle of the MCAT and are instructed to submit updates to it at least 24 hours in advance of need.

The Recovery Sequences.

At the outset, it is understood that it is practically impossible to 'quick fix' any catalog using some form of SUPERZAP. I am emphasizing this point because it tends to be attempted with alarming regularity. Please, inform you management now, if they don't already know, that an orderly approach to MCAT recovery is preferable to SUPERZAP.

The First Check.

Run LISTCAT ALL and compare to previous LISTCAT.
If the problem turns out to be minor, such as a missing alias or UCAT pointer, use TSO to catalog the required entry.
If the LISTCAT fails to complete properly, or shows a great many discrepancies, assume the catalog has permanent damage and initiate a full recovery sequence.

The Key Decision.

As long as MVS is still operable, (that is, the page and swap data sets are still functional), you can try a MCAT reload using the latest unloaded copy of the MCAT. This might save an IPL, which can turn out to be mighty important in an online system.
But, while you try this, the catalog might actually be severely damaged - limping along as it were. When the subsequent attempts to repair it online fail, you will have to initiate a major recovery sequence anyway, and the time spent in the futile REPRO logic will have been wasted.
IBM cannot advise you here. You must make this decision based on the number of users likely to be affected, the total time to do the recovery, the amount of money your company can lose due to an online system that is down, etc.
Experience by many users has indicated that if any doubt about the viability of the REPRO exists, the full recovery sequence is usually the shortest path to success.

A Warning.

Some customers have attempted to restore the volume containing MCAT online! In very limited cases, this has succeeded. (The checks made in IEHDASDR and similiar IBM products were bypassed by ZAPing the code in the utility.)
If you have read all of these notes, you should realize that MVS has built control blocks concerning the catalog in common storage, and will update the catalog only when MVS itself decides to do it. It short, to make this work, you would have to guarantee that the restored copy occupies the exact same tracks on the pack and that MVS did not decide to update MCAT in the middle of your utility with it bypassed integrity checking.

Unfortunately, this type of recovery is all too common with a customer who just converted to MVS from DOS, where 'everything was always the same.'
Finally, operators not trained in MVS can make these same mistakes, so be sure to put out the word, whatever you decide, about how to handle this problem on your system.

The Second Decision.

Once you make the decision that you are going to have to restore the volume that contains MCAT, your next decision is whether to do it in stand alone mode and then reIPL the main system, followed by subsequent REPROs LISTCATs, etc., or to IPL an alternate system, do the total repair on the alternate, and then reIPL the main system.
If you use the main system, you can obviously 'get going' faster. But you cannot execute commands to manipulate page and swap data sets that are open.
If you use the alternate system approach, the MCAT will become a UCAT in that system, and all objects in it can be manipulated since the page and swap data sets will not be open. However, two IPLs will be required. Also, it is assumed the alternate is up to date on VSAM maintenance.
In either case, the basic logic will be the same.

The Logic of the Recovery.

Restore a previous version of the MCAT. This is done to save the drudgery of recataloging all the aliases and nonVSAM entries that might be in MCAT.
Use REPRO to reload your latest unloaded copy of MCAT. This will update the alias and nonVSAM entries to their latest level.
Use LISTCAT to get a listing of what you now think is in the catalog, and compare it with you last LISTCAT. Correct any discrepancies using TSO or IDCAMS.
ReIPL the main system.

Alternate Techniques.

You can dispense with the dump/restore totally if you want to simply DEFINE a new catalog and then REPRO into it. This is fine, but the DEFINE requires you to be running, so this technique requires an alternate system.
It also eliminates the possibility of using the restore to 'get going' as described under the heading 'The Second Decisin' above.

You can obviously use a second system in your shop to recover the first, IF YOU HAVE A SECOND SYSTEM. This amounts to always having an 'alternate' available.

More Warnings.

One way for a disgruntled system programmer to 'get back' at you for being fired, etc., is to damage all your MCATS at once. This is out of the responsibility of VSAM - it's really your security problem. What have you done to guard against it?
There are also such things as 'acts of God' - fire, tornados, bombs, etc. In one case I know, the electrical contractor wired the wrong voltage to the computer room and the customer watched the machine literally disintegrate as it was hit with over 1000 volts.
What is your plan for 'acts of God'?