Undeleting Data: Part I

BY SAM GOLOB

Most of us have encountered a situation in which we accidentally deleted a pds member or a dataset. What does an application programmer do under when this happens? Call systems programming! They're the experts, and they'll restore the dataset from a backup (or they'll perform some "magic"). What do we do when it happens to us? Usually, we can do the same thing, except that we'll do it ourselves. But what if for some reasonthere aren't any backups? What recourse do we have?

Of course, this supposedly should never happen. We are the guardians of our installation's data. And we should administer the installation in such a way that there are many barriers between normalcy and disaster. If one thing goes wrong, then there is a contingency plan. If that contingency plan fails, then there's a second plan right behind it. Ideally, these contingency plans should be nested many levels deep, so that except under the most dire of circumstances, our shop can always recover from a mistake.

However, we and our management are fully responsible for setting up these plans and procedures in the first place. And it pays for us to become as knowledgeable as possible in recovery techniques. You never know when you'll need a technique or when you'll need a recovery tool in place, ready for action.

This month, I'll talk about some techniques that are usually in the category of "last resorts," but they don't have to be. Let's consider the idea of "undeleting," or restoring some data that was deleted by the operating system. Undeleting a pds member can often be done quickly with the right tools. However, undeleting an entire dataset is a completely different matter. It is usually very difficult to do, and to me it seems almost impossible to automate, because to undelete a dataset you must restore data that normally isn't saved. I'll discuss the topic of undeleting datasets briefly in this month's column, and continue the discussion in more depth next month. This month, I'll deal with general principles and how to restore deleted partitioned dataset members.

Once you've gotten rid of some disk data on your system, it's gone, right? Not necessarily. Sometimes the data itself still exists on disk, but only the pointers to the data were deleted. This happens when you delete a partitioned dataset member or a disk dataset.

PRINCIPLES BEHIND UNDELETING

Once you've gotten rid of some disk data on your system, it's gone, right? Not necessarily. Sometimes the data itself still exists on disk, but only the pointers to the data were deleted. This happens when you delete a partitioned dataset member or a disk dataset. Under many circumstances, the data is still there; only the pointers to the data have been wiped out. Theoretically, if you can restore the pointers to the data that is still there, you can restore actual access to the data. This is the concept of undeleting data.

Let's go deeper into this and ask the question, "How does the system find data on disk?" I'll answer this question in a general way. Once the system knows that the data is on a certain volume, usually from looking at the catalog or if the user has specified the particular volume serial, it will go to the volume id record on track 0, record 3 of that volume. Record 3 contains the volume id and the exact CCHHR (cylinder, head, and record) location of the header record (Format 4 DSCB) of the VTOC (Volume Table of Contents) for that volume. The VTOC contains records that point to the individual datasets. If the desired dataset happens to be a partitioned dataset, then the beginning of its data consists of another set of pointers, the pds directory. The pds directory contains the names of all the dataset members.

At a minimum, each pds directory entry contains the member name and the TTR location (relative track and record location from the beginning of the dataset) of the beginning of that member. The directory entry may contain more data such as SSI information, ISPF stats for source-type members, or load module attribute information. In any case, a dataset, or a member of a partitioned dataset, can be exactly located on the disk volume. Practically speaking, it's much easier to restore a deleted pds directory entry for a member than to restore all of the VTOC records necessary to "undelete" an entire dataset.

Now let's talk about "data integrity." What keeps the system from constantly overwriting valuable data that's on disk? Once data is written on a disk volume, its space locations, which are the disk extents, are marked as occupied, or "allocated." A new allocation of space will not reuse any space that's occupied — only space that's marked as "free." For performance reasons, if you think about it, it's obvious that the

dataset allocation component of the system should only be required to search for free space, and not have to sift through all the space on the pack that's occupied, before deciding where to put the new dataset. A new dataset should only go to free space, obviously. Therefore, the disk's VTOC system should have a quick way of finding its free space locations directly, without first sifting through the occupied space.

There's an old way and a new way of finding free space on a disk. The old way was from VTOC records directly. The new way uses another system dataset on the volume, known as the VTOC INDEX dataset. When you initialize a volume with IBM's utility ICKDSF, you optionally specify if you want the volume to be "indexed" and to have and use a VTOC INDEX dataset. Once the index dataset is created, it doesn't have to be used. Using the ICKDSF "OSVTOC" option, you can run a job to turn the use of the VTOC INDEX dataset off. Using the ICKDSF "IXVTOC" option, you can run another job to toggle the use of the VTOC INDEX dataset back on again. SMS-managed volumes are required to be indexed while they are being managed.

If the VTOC INDEX is not being used for a volume, then free space is determined by a chain of Format 5 VTOC records, known as free space records on the VTOC. If the VTOC INDEX is being used, then the index dataset maintains the current status of free disk extents on the pack. It's now obvious that whenever a new dataset gets allocated, its disk extents have to be removed from the pool of free disk extents on the volume. Conversely, when a dataset gets deleted, the VTOC system returns its extents back to the pool of free disk extents. This fact is very important for us. If you want to undelete a disk dataset, you have to ensure that no one else will write over the data areas where the dataset had been.

Before I move on to the easier task of restoring deleted partitioned dataset members (as opposed to restoring entire datasets), I have to mention the topic of VTOC consistency. It makes sense not to mark a disk extent in the VTOC as both being occupied and free at the same time. But as I described, the VTOC system keeps separate records of occupied space and free space. What if there is some kind of damage caused by a system crash, when the allocated extents and the free extents have gone out of sync? Where's the system safety net to guard against this?

To keep matters simple, I'll discuss the OSVTOC case, in which the VTOC INDEX dataset is not being used. In that case, all free space records are kept in a chain of Format 5 DSCB records in the VTOC itself. During any dataset allocation or deletion on the pack, there is a bit in the (Format 4) VTOC header, known as the DIRF bit, which is turned on. During the time when the DIRF bit is turned on, the system assumes that there are inconsistencies among the VTOC records. At the end of the allocation or deletion, after both "occupied" and "free" extent descriptions on the VTOC are adjusted to consistency, the DIRF bit in the VTOC header is turned off. The VTOC records are now assumed to be consistent between occupied space and free space.

Using FIXPDS can sometimes be a little tricky, especially if a source-type pds has written over a previous load module library.

Now what if a system crash occurs during a dataset allocation? When the system comes back up and you try to allocate a new dataset on the troubled volume, an allocation routine checks the DIRF bit first and sees that it's on. Allocation then calls a system program, known as the VTOC CONVERT ROUTINE, that goes through the entire VTOC and makes sure that the Format 5 free space records are consistent with the extents already marked as allocated before allowing any new allocations to take place. During that time, you'll see a console message: "VTOC CONVERT ROUTINE ENTERED, REASON DIRF." This safeguard ensures that new datasets will not be placed on top of current data after a crash.

So far, I've discussed how data is located on disk and how disk space is marked as either "allocated" or "free." The rest of this discussion will consist of two parts: undeleting pds members by restoring deleted pds directory entries, and undeleting entire datasets by reconstructing deleted VTOC entries. Since the latter topic is very difficult, I'll leave most of it for next month.

UNDELETING PDS MEMBERS

In truth, a partitioned dataset, or pds, is a sequential type dataset that has been modified to look like it is a collection of many datasets. Understanding that a partitioned dataset is "sort of sequential" is a key concept to know, when locating or restoring data in a pds. A good way to get a picture of a pds is to think that the data is stored sequentially, but it is located and kept track of by a kind of direct access. The component of the pds that tracks the individual files, or members, is known as the pds directory. My three-part series examined the details of pds directory structure (***Technical Support***, September through November 1991). These columns can be obtained from File 120 of the CBT MVS Tape, which is included on the NaSPA CD-ROM.

All new data written to a pds is written after the end of the existing data. For example, if you are editing some source-type data such as JCL or program source code, and you save the data, the data for the entire member is copied to a space after the end of all the other data in the dataset. After the new data is written, a pds directory entry is either created for a new member, or modified to point to the new data for a pre-existing member. In the case of an old pds member that was modified, after all of its data is copied to the end of the pds, its old data remains in place, not pointed to by any directory entry. Therefore the old data occupies "dead space" in the pds. The dead space is not recovered until a dataset "compress" is done, usually using the IEBCOPY program. "Undeleting" a pds member consists of re-creating a new pds directory entry to point to the previous dead space left by a deleted member.

The pds directory is physically located at the beginning of the first dataset extent and it almost always has a different DCB description than the data itself. Assembler programs that access a partitioned dataset's directory, as well as the data, have to have two different DCBs defined — one for the directory and one for the data that is in the members.

In restoring deleted pds members, bear in mind that there are enough automated or semi-automated tools for the job, so we won't have to directly zap a pds directory to do the restore. Some knowledge of actual directory structure will only be required if ISPF statistics, load module attributes, or SSI information needs to be adjusted in the restored member's directory entry; but we also have tools for that. Available utilities

that use the STOW macro to make a new directory entry that points to the deleted data are quite plentiful. Our job will consist of learning how to use these tools. This is the main reason why undeleting a pds member is so much easier than undeleting an entire dataset. To my knowledge, there are no automated tools available to undelete an entire dataset.

POWERFUL TOOLS FOR UNDELETING

What are some pds member undeletion tools and how can we get them? One of the most powerful tools is the free "PDS" TSO command processor that is on File 182 of the CBT MVS Utilities Tape, a huge independently produced collection of systems programmer utilities that can be obtained (among other places) through the NaSPA office. Another, very different tool, is the FIXPDS program from File 036 of the CBT Tape. A third tool is the PDSGAS batch program from File 316 of the CBT Tape. The CBT Tape also has other programs for this purpose scattered throughout its files.

The easiest tool to use, in my opinion, but not the best, is the program called PDSGAS. PDSGAS is a batch program that scans the data part of a partitioned dataset from the beginning. If any data is found that doesn't have a directory entry pointing to it, PDSGAS creates members named $$000001, $$000002, and so forth to point to the data in these "dead spaces." If those names already exist, PDSGAS changes the new names slightly. PDSGAS can do a minimal restore for source-type partitioned datasets only. No ISPF statistics or SSI values are created for the newly made members. PDSGAS also can't recreate the special directory entries necessary to point to load module pds members.

The next tool to learn about is the free PDS command processor, now at Version 8.5, from File 182 of the CBT Tape. PDS is an extremely versatile program package. There is a vendor-supported version of PDS called STARTOOL (from SERENA International) that is far better than the free version, but for this job, either product will do. When restoring deleted pds members, the PDS product is extremely smart. PDS can tell the difference between source-type members and load modules, and it can supply ISPF statistics for the source-type members that it restores. For load modules, once they are restored using PDS's "RESTORE" subcommand, the load module attributes, such as reentrancy, APF authorized, special entry points, etc., can be readjusted using PDS's "ATTRIBUTE" and "ALIAS" subcommands. The PDS command statements necessary to get a similar (but much more intelligent) result to PDSGAS, are:

```
RESTORE $$ REPEAT NOPROMPT
```

The PDS package has an extensive HELP member where you can find out many more details.

Finally, FIXDS is a completely different tool for pds member restoration and is on File 036 of the CBT Tape. Instead of starting its scan for deleted member data at the beginning of the partitioned dataset, FIXPDS starts at the end of the extents for the dataset and goes backwards. Each piece of "dead space" found by FIXPDS is ISPF-browsed for the user, and the user is given the opportunity to decide whether or not to stow a new member name for that data. FIXPDS ends when it encounters the beginning of the pds.

Using FIXPDS can sometimes be a little tricky, especially if a source-type pds has written over a previous load module library. Since FIXPDS does an ISPF browse, it uses the source dataset's DCB information such as its LRECL and BLKSIZE. When it goes to the end of the extents and sees old load modules that are now dead space, it usually can't browse them because their blocks are too big. Therefore, to use FIXPDS on a dataset like that, you sometimes have to temporarily alter (with a tool like CDSCB from File 300 of the CBT Tape) its DCB information to LRECL=0, BLKSIZE=32760, RECFM=U, look at the "far out" data, stow what member names you want, and then change the DCB information back to what it was. This is tedious to do, and for member restores in general, I'd say that the PDS package is the best tool to use.

Well, I hope this month's column has been instructive! Next month, I'll try to unravel the tricky concepts behind restoring entire deleted datasets. Good luck, and see you then. *ts*

Sam Golob is a senior systems programmer. He can be reached at sbgolob@aol.com or sbgolob@ibm.net. Documentation about the CBT MVS Tapes can be found on the web at http://members.aol.com/cbttape.