Undeleting Data: Part II

BY SAM GOLOB

Last month's column examined how to "undelete" data that had been deleted from disk. I asked the question: "If data is deleted from disk and isn't there any more, how is it possible to get it back?" The answer is that often the process of deleting disk data only involves removing a pointer to the data, as opposed to wiping the data off the disk entirely. If we can properly restore all the pointers to the data, then we can recover the data. The trick is, do we truly know the way the system is pointing to the data, across changes in the operating system and in data management from release to release and from device to device?

Life has become more complicated. System Managed Storage has introduced additional intricacies into how disk data is kept track of. Additionally, several flavors of RAID devices have partly or completely replaced conventional DASD in many installations. These devices differ greatly in the internals of their workings, and we have to know whether or not they will logically behave like "real DASD." Our approach, when faced with the tough job of recovering irreplaceable data that's "still there," is to be clever, simplify the situation as much as possible, keep a clear head and to stick to basics.

This month's discussion concentrates on how to recover data from a disk dataset that was "deleted" by the system. Last month, I examined many basic principles of how data is located on disk, as well as a few tools that are capable of restoring deleted pds members. When restoring a pds member, bear in mind that the dataset itself has not disappeared, just a directory entry that points to the former location of a member has been wiped out. A pds directory entry contains relatively little new data, minimally a member name and a relative TTR (track and record) location from the beginning of the dataset. This can be automatically scanned for by a program and filled in. Since a new pds member is always written at the end of the previous data, there is no problem of keeping track of "free space" and preventing a new member from writing over it. Now, we're faced with what happens when the entire dataset, pds or otherwise, is gone. What can we do to effectively get it back? All the extent information is gone. All the DCB information is gone. We haven't saved the VTOC information. In short, recovering a deleted dataset is a far tougher problem to tackle.

TOOLS WE WILL NEED

Generally, I write about tools that are available for everybody to use, no matter how "low budget" their shop. I'm glad that these tools are available, and I'd like to thank all the people who wrote, fixed and contributed them. Most of these tools (at least the ones that don't come from IBM) can be found on the CBT MVS Utilities Tape. This is a huge, free, independently produced collection of "systems programmer tools" that can be obtained from NaSPA (among other places). There is also a CBT Overflow Tape that contains a lot of additional public software. Version 416 of the CBT MVS Tape is included on the newest NaSPA CD-ROM; however, the tapes are usually updated several times between editions of the CD-ROM. Current information about the CBT Tapes is maintained at web site http://members.aol.com/cbttape.

In order to attempt an undelete of a disk dataset, we'll need to set up some tools from the CBT Tape (or others of equivalent functionality, if available). For relative simplicity, I'll limit this discussion to specifically attempting to undelete non-VSAM datasets.

First, we'll need a disk mapping program that shows track extents. One example is the IEHMAP program from File 083 of the CBT Tape. Similar programs are called MAPDISK. The Fullscreen ZAP TSO command (from File 134; load module on File 135) will have to be run in authorized mode. To learn how to do "non-invasive private TSO authorization for sysprogs," see Files 185 and 186 of the CBT Tape. If module IKJTABLS is run from an authorized STEPLIB it overrides (the public) settings in IKJTSOnn of PARMLIB.

Other helpful tools include the PDS TSO command package from File 182. If you're licensed for PDS's successor, STARTOOL from SERENA International, that will be useful; however, for this job, STARTOOL is not necessary. STARTOOL has a facility to format (and change) all the Format 1 DSCB fields once a working Format 1 DSCB has been created. Of course, we'll use ISPF. The REVIEW TSO browser from File 134 (load module on File 135) will also be helpful.

WORKING ON A PROBLEM

We all shudder to think about this scenario: dataset deleted, no backup. It's a pds (source or load module). For simplicity, it's on a seldom-used non-SMS pack. To curtail usage of the pack, we might try varying it offline to all systems except a test MVS system or using some other clever trick as necessary. If the data was on an SMS-controlled pack, the pack has to be removed from SMS control. See the end of Figure 1 for sample JCL.

Now, the first thing we have to do is to disable the VTOC index, if there is one. For that, we'll have to run an ICKDSF OSVTOC job (also see Figure 1). If the VTOC is nearly full, we'll have to make sure there's room for a few extra Format 5

Figure 1: Jobs You Will Need: A Quick Reference

These are jobs to un-index the VTOC and re-index the VTOC of a pack. It is assumed that the pack was originally indexed, and we do not want to use the index dataset while zapping VTOC entries. When the VTOC is un-indexed, only the VTOC itself has meaningful information about the datasets on a pack. The index is then not used. Re-indexing the pack will have the effect of reorganizing the VTOC index and recalculating the pack's free space. At the bottom of the figure is a DFDSS job to remove an SMS volume from SMS control. After the DFDSS job is complete, you have to remove the volume from the storage group. Then you can access the datasets (and curtail activity on that volume at the same time).

```
//*              * * *     O S V T O C     * * *
//SAGOLOBO JOB   (rest of job card)
//*******************************************************************//
//*  NOTE..... THIS WILL ASK OPERS FOR A REPLY - YOU NEED A "U" IF OK.
//*******************************************************************//
//*      CONVERT IXVTOC TO OSVTOC:
//****************************************//
//S1     EXEC PGM=ICKDSF,TIME=1439
//SYSPRINT  DD SYSOUT=*
//VPDKZ01 DD VOL=SER=PDKZ01,UNIT=SYSDA,DISP=SHR, <=== CHANGE ALL
//   DSN=SYS1.VTOCIX.VPDKZ01         <=== OCCURRENCES OF VOLSER NAME.
   BUILDIX DDNAME(VPDKZ01) OSVTOC
/*
//

//*              * * *     I X V T O C     * * *
//*******************************************************************//
//*    NOTE..... THIS WILL ASK OPERS FOR A REPLY.  "U" IF OK. ****
//*******************************************************************//
//*        REBUILD INDEXED VTOC:
//****************************************//
//START EXEC PGM=ICKDSF,TIME=1439
//SYSPRINT  DD SYSOUT=*
//VPDKZ01 DD VOL=SER=PDKZ01,UNIT=SYSDA,DISP=SHR, <=== CHANGE ALL
//   DSN=SYS1.VTOCIX.VPDKZ01  <=== OCCURRENCES OF VOLSER NAME TO YOURS
   BUILDIX DDNAME(VPDKZ01) IXVTOC
/*
//

//****************************************************************//
//*    USING THE ADRDSSU CONVERTV COMMAND TO GO FROM SMS.        *//
//*    THIS EXAMPLE CONVERTS A VOLUME TO NON-SMS-MANAGED.        *//
//*      see DFDSSdss STORAGE ADMINISTRATION REFERENCE           *//
//****************************************************************//
//*
//STEP1     EXEC   PGM=ADRDSSU
//SYSPRINT  DD     SYSOUT=*
//SYSIN     DD     *
          CONVERTV -
                DYNAM(SMS034) -
                NONSMS
/*
//
```

(free space) entries by moving a few datasets to another pack. Once the VTOC is non-indexed we can start work.

The idea is to take a blank VTOC entry that's all hex zeros (Format 0 entry) and create a Format 1 VTOC entry that points to the correct data extents and behaves "authentically." After this work is done, the pack "free space" must be adjusted so the new "dataset" we've created will have extents that are marked as allocated. Finally, we'll use ISPF 3.2 and other ISPF functions to look at the new dataset and see if it behaves as it should. Then we'll make minor adjustments as necessary and copy the data to another pack if we can.

Our next task is to figure out where the data was. Initially, we have to find the first extent. For a pds, this will be a directory, and if we see it, we should be able to identify the member names. If the member names look like those in the deleted dataset, we've made headway. Deleted datasets leave unoccupied data extents on the disk pack. Next, we have to map the disk pack with a tool that has an absolute track mapper (we'll use IEHMAP from File 083 of the CBT Tape) and examine all of the empty extents that look like they are the proper size. That is, if we know that the deleted dataset had a 300-track first extent, we'll look in the IEHMAP report for an unoccupied extent of 300 tracks to search first. Here's where you have to use your head!

After we've picked some likely extents where the data might have been, we need a tool to examine the data that's there. It's not every systems programmer who has a tool to look at unallocated data on a disk pack! But we have one — it's Greg Price's version of UCLA Fullscreen ZAP — running authorized and in FULLVOL mode. We can examine all the data in the entire disk pack with Fullscreen ZAP and we can go to absolute cylinder, track, and record locations.

Often the process of deleting disk data only involves removing a pointer to the data, as opposed to wiping the data off the disk entirely. If we can properly restore all the pointers to the data, then we can recover the data.

We will invoke ZAP in FULLVOL mode by saying: TSO ZAP 'FORMAT4.DSCB' VOL(volser) FULLVOL. This takes us to Track 0, Record 1 at the very beginning of the pack. Next, we invoke the "R" subcommand (go to the next record) of ZAP twice to bump up to Record 3. This will show the pack volser, so we're sure it's the right pack. Actually, the pack id is displayed at the bottom of the ZAP screen. Then we use the ABS (go to absolute CCHHR location) subcommand of ZAP to get directly to the CCHHR location on the pack. When using ABS, you may have to play with leading zeros as follows: Suppose that IEHMAP shows a likely gap starting at location 05E0.0007. That's cylinder number X'05E0', track 7 (really one more — numbering starts from zero). To look there, we enter ABS0005E00007, give or take a leading 0. If the old data was there, we should be looking at Record 1 on the track, an obvious pds directory, so that we can read the names of the members. Otherwise, if it's the wrong place, we'll try the next likely gap (shown by IEHMAP) until we hit paydirt. This could take a lot of patience.

Figure 2: STARTOOL'S Layout of a Format 1 DSCB for a PDS

This figure will help us visualize the layout of a 140-byte VTOC Dataset Description entry (Format 1). The data description can also be found by assembling the macro IECSDSL1 1 from SYS1.MODGEN, but STARTOOL shows, in addition, what the data actually looks like. Figure 3 shows how Fullscreen ZAP views this same data. The dataset is 80-byte, fixed blocked card-image format.

```
 ----------- FIXPDS: Format 1 DSCB Modification ------------------------------
 OPTION  ===>

 Current - DSN=SAGOLOB.CBT.EDIT,VOL=SER=PRIM12  MEM=A  -----------------------
  DSCB1 at 0000000519
```

Caution: Many of the fields of the Format 1 DSCB can not be changed without compromising dataset integrity. For more information using field level help, place the cursor on any of these items and press HELP.

```
OFF,LEN     LABEL          HEX VALUE                DESCRIPTION
(0,44)   DS1DSNAM  ===>  SAGOLOB.CBT.EDIT
(2C,1)   DS1FMTID  ===>  F1               Format identifier
(2D,6)   DS1DSSN   ===>  D7D9C9D4F1F2     Data set serial name
(33,2)   DS1VOLSQ  ===>  0001             Volume sequence number
(35,3)   DS1CREDT  ===>  6100D1           Creation date
(38,3)   DS1EXPDT  ===>  000000           Expiration date
(3B,1)   DS1NOEPV  ===>  07               Number of extents on volume
(3C,1)   DS1NOBDB  ===>  00               Number of bytes used in last directory
(3D,1)             ===>  00               Reserved
(3E,6)   DS1SYSCD  ===>  C9C2D4D6E2E5     System code
(44,7)             ===>  E2F24040404040   System code (last 7 characters)
(4B,3)   DS1REFD   ===>  62006D           Date last referenced
(4E,1)   DS1SMSFG  ===>  A0               System managed storage indicators
(4F,1)   DS1SCXTF  ===>  00               Secondary space extension flag
(50,2)   DS1SCXTV  ===>  0000             Secondary space extension value
(52,2)   DS1DSORG  ===>  0200             Data set organization
(54,1)   DS1RECFM  ===>  90               Record format
(55,1)   DS1OPTCD  ===>  00               Option code
(56,2)   DS1BLKL   ===>  6D10             Block length
(58,2)   DS1LRECL  ===>  0050             Logical record length
(5A,1)   DS1KEYL   ===>  00               Key length
(5B,2)   DS1RKP    ===>  0000             Relative key position
(5D,1)   DS1DSIND  ===>  82               Data set indicator flags
(5E,4)   DS1SCALO  ===>  8000005B         Secondary allocation type and amount
(62,3)   DS1LSTAR  ===>  019002           TTR of last used track and block of data
(65,2)   DS1TRBAL  ===>  CFFC             Bytes remaining on last track used
(67,2)             ===>  0000             Reserved
(69,10)  DS1EXT1   ===>  010005E0000705F40004  Extent 1 in XX00CCCCHHHHCCCCHHHH
(73,10)  DS1EXT2   ===>  0101077C000E077D000E  Extent 2 in XX01CCCCHHHHCCCCHHHH
(7D,10)  DS1EXT3   ===>  0102025C000C025D000E  Extent 3 in XX02CCCCHHHHCCCCHHHH
(87,6)   DS1PTRDS  ===>  0000000506       CCHHR of any associated Format 2 or 3 DSCB
```

Now that we know where the beginning of the data is, we have to build a VTOC pointer to it. This is the tricky part. We assume that the VTOC index dataset has been disabled by job OSVTOC (Figure 1), and therefore, all dataset information is in the VTOC itself. In the ZAP command (invoked as before), enter the subcommand VTOCDS4. This will get us to the first record (Format 4 — VTOC header) in the VTOC. We start stepping forward, advancing one record at a time, using the "R" subcommand repeatedly, until we come to a VTOC record that's all zeros. Look at the bottom of the screen and make a note of the TTR location of this record. We're going to use ZAP to turn this record into a Format 1 VTOC record (DSCB) that points to our deleted data.

When creating a new Format 1 DSCB we have to be very careful that the system believes us — every field has to be correctly entered with no mistakes. You can find out the format of a Format 1 DSCB by going to SYS1.MODGEN and assembling the macro IECSDSL1 with keyword 1 following it. Each VTOC entry is a keyed record — a 44-byte key at the beginning, followed by a 96-byte data area — 140 bytes in all. The records in a VTOC are unblocked. For some formats, the key is abbreviated, not all of it is used. For the Format 1 DSCB, all of the key, the first 44 bytes, is the dataset name and blanks (X'40') have to pad to the end of the 44 bytes.

Since I have limited space to describe a big job, I've included Figures 2 and 3 as models that detail the layout of a Format 1 DSCB for an FB-80 partitioned dataset. Figure 2 was created by the FIXPDS subcommand of the vendor product STARTOOL, which formats the VTOC entry of the dataset that is pointed to by STARTOOL. Not everybody has STARTOOL, so we'll use Fullscreen ZAP and the DSCB 1 DSECT description as our guide. Figure 3 shows how the same VTOC record appears to Fullscreen ZAP.

You should create your own model "real" VTOC entry as well. For example, if the deleted dataset was sequential, look at the VTOC entry for a similar sequential dataset as your model. If the deleted dataset was a load library, look at the VTOC entry for a similar load library as your model. You can do this with Fullscreen ZAP itself. The Fullscreen ZAP tutorial (subcommand "?") will show you how to print the current record or several records. Point Fullscreen ZAP to the VTOC entry for your model dataset (one that's similar to the deleted one) and print out its entire record.

Now that we're "ready," we have to point to the beginning of the blank VTOC record (the Format 0) that we've chosen and start filling in the fields. For this, we use the "S" subcommand of Fullscreen ZAP to change the data bytes of the record up to 8 bytes at a time. Then we (hold our breath and) make the changes permanent with 'ZAP' subcommands as we go. When all the fields are filled in, we try and access the data using ISPF 3.2 with dataset and volume specification first to see if ISPF can find the dataset at all. After fixing any obvious errors by re-zapping VTOC fields we can try to ISPF BROWSE the dataset to see if we have the data. Finally, a good next step would be to try and copy the data to a new dataset on another pack. Don't use IEBCOPY for this, but rather try ISPF 3.3. ISPF 3.3 will disturb the "from" data less than IEBCOPY. I've had nightmares using IEBCOPY here.

The new dataset is of course uncataloged. I'd try and copy the data off and then re-delete the dataset. This is probably the safest practice. You may have to find additional extents for the dataset, and you'll have to repeat the previous IEHMAP and Fullscreen ZAP "ABS" searches patiently. Then you have to fill in "second extent" and "third extent" fields in the Format 1 VTOC entry. For more than three extents, you'll have to find another blank entry to make a

Figure 3: Fullscreen ZAP Representation of a Format 1 DSCB Record

This might serve as a model for constructing a new Format 1 DSCB to point to our deleted dataset. This is the same record that was formatted by STARTOOL in Figure 2.

```
                               Z  A  P

ENTER VALID COMMAND ABOVE OR ? FOR HELP          VERSION=2.3G 17AUG92
F'CBT.EDIT'

00000  E2C1  C7D6  D3D6  C24B >C3C2  E34B  C5C4  C9E3   | SAGOLOB.CBT.EDIT |
00010  4040  4040  4040  4040  4040  4040  4040  4040   |                  |
00020  4040  4040  4040  4040  4040  4040  F1D7  D9C9   |             1PRI |
00030  D4F1  F200  0161  00D1  0000  0007  0000  C9C2   | M12../.J......IB |
00040  D4D6  E2E5  E2F2  4040  4040  4062  006D  A000   | MOSVS2      .._.. |
00050  0000  0200  9000  6D10  0050  0000  0082  8000   | ......_..&...b.. |
00060  005B  0190  02CF  FC00  0001  0005  E000  0705   | .$..........\... |
00070  F400  0401  0107  7C00  0E07  7D00  0E01  0202   | 4.....@...'..... |
00080  5C00  0C02  5D00  0E00  0000  0506               | *...)........... |


                  *****  SCAN MATCH  *****
OFF: 0008 (    8)   ADDR: 00008 (    8) DSN: VTOC FOR PRIM12
LEN: 008C (   140) BASE: 00000 (    0) CCHHR: 0000000519 TTR:  000419
```

Format 3 DSCB, point to the Format 3 from the Format 1 (the Fullscreen ZAP TTR display may help you), and you have to correctly make a new Format 3 entry as well. A Format 3 DSCB will hold up to 13 additional extent descriptions, from the fourth extent to the sixteenth. Assemble IECSDSL1 3, look at a model, and keep on trying until all the missing data has been recovered. Then copy it off to save it. I talked last month about how to re-adjust the pack's free space.

I hope this month's column has enlightened you as to how tough it is to recover a deleted dataset and how important it is to not get to this point. If you can devise a way to keep records of all your VTOC entries for all your packs, the potential job of recovering will be much easier. Even if you can IEHMAP all your packs once a week (IEHMAP is very quick), this will be of invaluable help. Meanwhile, keep your thinking caps on! See you next month. ts

NaSPA member Sam Golob is a senior systems programmer. He also participates in library tours and book signings with his wife, author Courtney Taylor. Sam can be contacted 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.