VSAM Tutorial

Virtual **S**torage **A**ccess **M**ethod - VSAM - is a data management system introduced by IBM in the 1970s as part of the OS/VS1 and OS/VS2 operating systems. Although there are still datasets that are best managed with the several other (non-VSAM) data management methods, VSAM is a major component of modern IBM operating systems. Since MVS 3.8 is one of those operating systems, I thought it might be useful to other Hercules' users to set down some basic information about VSAM.

I have divided the material presented here into two main segments -

In the first segment, I will try to provide a simple description of the components of VSAM, with the goal of introducing VSAM to those who have not had practical experience with it. I don't want to write a textbook, as I have several of those in my own library and they can be quite dry and boring. But, it is my perception that quite a few people are coming into the Hercules (and MVS) community who have not had any formal exposure to this type of material and I think there may be some positive benefit to my efforts.

In the second segment, I will try to cover most, if not all, of the functions provided by Access Method Services. Access Method Services is the single, general-purpose utility that is used to manipulate VSAM components by both Systems and Applications Programmers. If you are more interested in the "how to" rather than the "why", this second segment is probably where you will want to begin reading.

There is also an index at the bottom of the page to allow you to quickly locate the section describing a particular AMS function.

Concepts and Facilities

VSAM was, by several accounts, intended to replace all of the earlier data management systems in use by IBM's operating systems. Conventional (non-VSAM) access methods generally provide only a single type of dataset organization. VSAM provides three:

- **K**ey **S**equenced **D**ata **S**et (KSDS), where each record is identified for access by specifying its key value - a sequence of characters embedded in each data record which uniquely identify that record from all other records in the dataset. KSDS datasets are similar to Indexed Sequential Access Method (ISAM) datasets, with many of the same characteristics, but also having distinct advantages over ISAM.
- **E**ntry **S**equenced **D**ata **S**et (ESDS), where each record is identified for access by specifying its physical location - the byte address of the first data byte of each record in relationship to the beginning of the dataset. ESDS datasets are similar

to Basic Sequential Access Methid (BSAM) or Queued Sequential Access Method (QSAM) datasets.

- **R**elative **R**ecord **D**ata **S**et (RRDS), where each record is identified for access by specifying its record number - the sequence number relative to the first record in the dataset. RRDS datasets are similar to Basic Direct Access Method (BDAM) datasets.

VSAM datasets are frequently referred to as *clusters*. A KSDS cluster consists of two physical parts, an *index* component, and a *data* component. ESDS and RRDS clusters consist of only a single component, the *data* component.

KSDS Cluster Components

Each record in the data component of a KSDS cluster contains a key field, which must be the same number of characters and occur in the same relative position in each record. The records are stored in the data component in logical sequence based upon their key field value. The index component of the KSDS cluster contains the list of key values for the records in the cluster with pointers to the corresponding records in the data component. The records in a KSDS may be accessed sequentially, in order by key value, or directly, by supplying the key value of the desired record. The records of a KSDS cluster may be fixed length or variable length. Records may be added or deleted at any point within a KSDS cluster, and the affected record is inserted or removed, and the surrounding records will be reorganized as required to maintain the correct logical sequence.

ESDS Cluster Components

The records in an ESDS cluster are stored in the order in which they are entered into the dataset. Each record is referenced by its *relative byte address* (RBA). In an ESDS dataset of 100 byte records, the RBA of the first record is 0, the RBA of the second record is 100, the RBA of the third record is 200, etc. The records in an ESDS may be accessed sequentially, in order by RBA value, or directly, by supplying the RBA of the desired record. The records of an ESDS cluster may be fixed length or variable length. Records may not be deleted from an ESDS cluster, and they may only be added (appended) to the end of the dataset, following records previously written.

RRDS Cluster Components

The records in an RRDS cluster are stored in fixed length slots. Each record is referenced by the number of its slot, which is a number varying from 1 to the maximum number of records which may be contained in the dataset. The records in an RRDS cluster may be accessed sequentially, in relative record number order, or directly, by supplying the relative record number of the desired record. The records of an RRDS cluster must be of fixed length. Records may be added to an RRDS cluster by writing a new record's data into an empty slot, and records may be deleted from an RRDS cluster, thereby leaving an empty slot where the record that was deleted was previously stored.

Control Intervals

In non-VSAM data management methods, the unit of data that is moved between memory and the storage device is defined by the block. In VSAM, the unit of data that is transferred in each physical I/O operation is defined as a *control interval*. A control interval contains *records*, *control information*, and (in the case of KSDS clusters) possibly *free space* which may later be used to contain inserted records.

When a VSAM dataset is loaded, control intervals are created and records are written into them. With KSDS clusters, the entire control interval is usually not filled. Some percentage of free space is left available for expansion. With ESDS clusters, each control interval is completely filled before records are written into the next control interval in sequence. With RRDS clusters, control intervals are filled with fixed-length slots, each containing either an active record or a dummy record. Slots containing dummy records are available for use when new records are added to the dataset.

Control Areas

Control intervals are grouped together into *control areas*. The rules used for filling and writing control areas are similar to those which apply for control intervals. For ESDS and RRDS clusters, control areas are filled with control intervals that contain records. For KSDS clusters, some of the control intervals in each control area may consist entirely of free space that can be used for dataset expansion.

VSAM Catalogs

When a non-VSAM dataset is created, the user has the *option*, by means of the DISP=(,CATLG) JCL entry, of creating a catalog entry for the dataset. The catalog keeps track of the unit and volume on which the dataset resides and can be used for later retrieval of the dataset. With VSAM datasets, creation of a catalog entry to record the unit and volume, as well as many other characteristics of the dataset, is not optional.

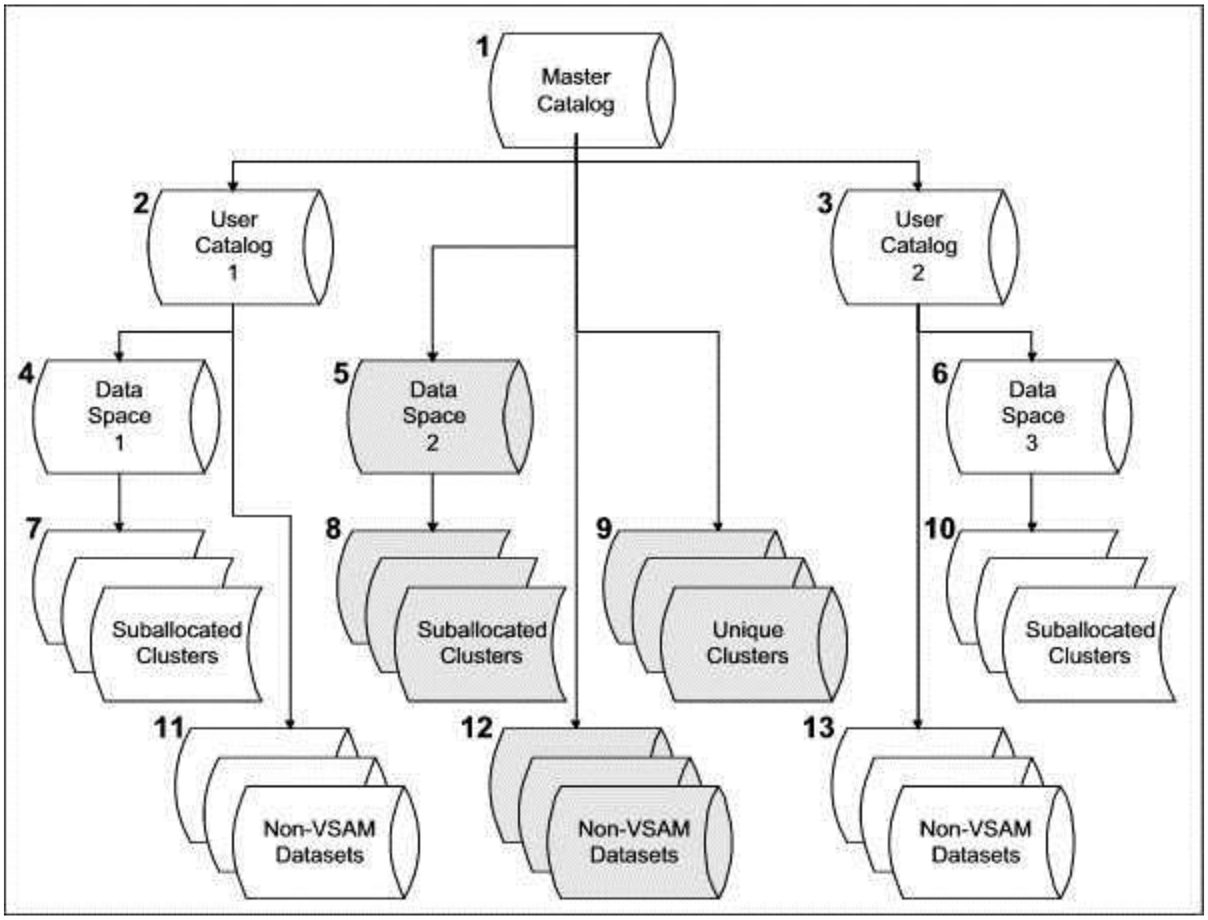
Prior to VSAM, catalog entries for non-VSAM datasets were contained in OS CVOLS (operating system control volumes). VSAM maintains its own catalog, which is itself a KSDS cluster, into which catalog entries describing VSAM clusters are recorded. The same VSAM catalog may also be used to contain the catalog entries for non-VSAM datasets.

Later releases of OS/390, the operating system into which MVS evolved, and z/OS, the current incarnation of MVS-OS/390, use yet another catalog system - the Integrated Catalog Facility. On the latest versions of OS/390 and z/OS, ICF catalogs are the only type of catalogs supported.

For MVS 3.8j, the relevant catalog system is the VSAM catalog, which is where information for both VSAM and non-VSAM datasets is recorded.

Catalogs, Data Spaces, and Clusters - Oh My!

There is what may seem at first a complex and baffling set of relationships between the underlying components of VSAM. I will begin with a graphic representation of the components and their relationships to one another, and then describe the rules governing the relationships.



Master Catalog

Every system that uses VSAM has one, and only one, *master catalog*. The master catalog contains entries about system datasets and VSAM structures used to manage the operation of VSAM. It is possible for any dataset (VSAM or non-VSAM) to be cataloged in the master catalog, but that is rarely allowed in well managed systems. In most computer systems, the Systems Programming staff will have created user catalogs, which are cataloged in the master catalog; all other users of the computer system will only be allowed to catalog datasets in those user catalogs. In the chart, the objects which have been shaded gray are the objects (excluding user catalogs) which are cataloged in the master catalog. In a typical system, these objects would all be system datasets, such as the system libraries (non-VSAM datasets) and page datasets.

The master catalog is created during the System Generation process and usually resides on the System Residence volume. The master catalog "owns" all other VSAM resources in a computer system, and this is denoted by the position of the master catalog (#1) in the chart. To quote a fairy tale that was popular in the 1970s that was used to describe the relationship of VSAM components, the master catalog is the "VSAM King".

User Catalogs

A *user catalog* is a catalog created to contain entries about application specific datasets. The information defining a user catalog is stored into a catalog entry in the master catalog. A production system might have any number of user catalogs, with the datasets cataloged in a specific user catalog related by application type. There are two user catalogs shown in the chart (#2 and #3).

Catalog - Volume Ownership

When a given direct access storage volume contains a VSAM catalog (either the master catalog or a user catalog), the catalog must be the first VSAM object stored on that volume. **A VSAM catalog *owns* the volume on which it resides.** All the VSAM objects that are defined on a volume containing a VSAM catalog must be cataloged in the catalog residing on that volume. A catalog can also own other volumes; however, those volumes cannot also contain other VSAM catalogs. When the first VSAM object on a volume that does not contain a VSAM catalog is defined, that volume becomes the property of the catalog that contains its catalog entry.

VSAM Data Space

Before VSAM clusters can be created on a volume, one or more data spaces must be created. A data space is an area of the direct access storage device that is exclusively allocated for VSAM use. That is, the area occupied by the data space is recorded in the Volume Table of Contents (VTOC) of the volume as allocated to a dataset, so that the space will not be available for allocation to any other use, either VSAM or non-VSAM. There are three data spaces shown in the chart (#4, #5, and #6).

Actually, when either the master catalog or a user catalog is defined, VSAM creates a data space to hold the user catalog entries, allocating the amount of space specified for the user catalog from the space available on the volume as a data space which will be completely allocated to the catalog..

The name of the files that are recorded in the VTOC for space allocated to catalogs and data spaces is generated by VSAM. However, they can be easily recognized for what they contain from the high level qualifier of the generated name. For catalogs (both master and user), the high level qualifier is Z9999992. For data spaces, the high level qualifier is Z9999994.

Unique Clusters

It is possible to create VSAM clusters out of unallocated space on direct access storage. This type of cluster has a designation of UNIQUE and essentially consists of a separate data space which is utilized completely by the cluster created within it. From a data management viewpoint, it is not a good idea to create unique VSAM clusters, although in some cases there are system datasets which are created in this manner. There is an indication of this type of cluster allocation on the chart (#9).

The most frequent manner of creating VSAM clusters is to suballocate the space required for the cluster's records from available space in a previously defined data space. Suballocated clusters are indicated on the chart for both system and user datasets (#7, #8, and #10).

For all VSAM objects **except** suballocated clusters, there is an entry placed in the VTOC of the direct access storage device volume on which the object resides. The entry name generated by VSAM for these objects is usually not mnemonic enough to visually indicate the contents of the VSAM object.

Non-VSAM Datasets

In addition to VSAM objects, non-VSAM datasets (residing on both tape and direct access storage) may have entries in both the master catalog and user catalogs. As with VSAM objects, it is best if only system datasets are cataloged in the master catalog. Since the main function of cataloging non-VSAM datasets is to retain Unit and Volume Serial information, the amount of information stored for a non-VSAM dataset is minimal compared to the information stored for a VSAM object. Non-VSAM objects are indicated on the chart as #11, #12, and #13.

Want to know more details? VSAM: Concepts, Programming, and Design by Jay Ranade, Macmillan, 1986, ISBN 0029486300 is probably the definitive volume. I have seen a copy, but never got around to owning one of my own. ABE has some copies if you are interested. The follow on to that book was VSAM: Performance, Design, and Fine Tuning, and it has really served me well in years gone by.

Access Method Services

Pre-VSAM data management required many utility programs for housekeeping -

- IEBGENER to copy the contents of sequential datasets
- IEHMOVE and IEBCOPY to copy, move, reorganize, expand, backup and restore the contents of partitioned dataset
- IEBISAM to load, backup, restore, and reorganize the contents of ISAM datasets

to name a few. However, there is only a single utility for managing all of the housekeeping needs of VSAM -

IDCAMS

which is also known by the functionality it provides - Access Method Services - or, simply, AMS.

The basic JCL to run the utility is:

```
//IDCAMS   JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *
        /* UTILITY COMMAND STATEMENTS */
/*
//
```

Some of the functions require additional DD statements when they are referenced by the AMS command statements, but SYSPRINT and SYSIN are the only absolute requirements.

There are a number of ways to organize the many functions that AMS provides. I will present functions as they might be required on a newly generated system. For example, the first task would be to set up user catalogs and data space to accommodate the creation of VSAM clusters. At the end of this page I will place an Index table with links to bookmarks so that anyone looking for a specific function can find it more easily.

It is not my intention to cover every single function provided by AMS, but what I include here should cover the needs of 99% of users, at least as it relates to MVS 3.8j. For any needs I may have overlooked, there are always the IBM manuals available online. All of the example jobstreams on this page are contained in a single archive that is available to download from my site: vsjcl.tgz.

Statement Syntax

The format of AMS commands is basically free form, and resembles REXX or PL/1. The default statement margins are positions 2 through 72. Any command statement may be continued from one line to the next by following the last parameter in a line with a hyphen (-). A value may be continued by immediately following it with a plus sign (+). A comment may be embedded in the command statements by enclosing the comment characters with /* and */. Blank lines may be included at any point before, interspersed with, or following AMS commands. Positional parameters are not optional and must precede all keyword parameters. Keyword parameters can stand alone or they may have an associated set of values or a subparameter list enclosed in parentheses. Parameters, subparamenters, and values are separated from one another by spaces, commas, or comment blocks.

Modal Commands

It is possible to include AMS commands to perform more than one function in a single execution of the IDCAMS utility. Therefore, AMS sets a return code following the execution of each command, and also maintains a maximum return code value for each execution. AMS commands are provided to interrogate these return codes and conditionally execute command statements based on the return code set during the execution of prior commands.

The return codes set by AMS can be interpreted as -

- 0 - Normal Completion - the functional command completed its processing successfully
- 4 - Minor Error - processing is able to continue, but a minor error occurred, causing a warming message to be issued
- 8 - Major Error - processing is able to continue, but a more severe error occurred, causing major command specifications to be bypassed
- 12 - Logical Error - generally, inconsistent parameters are specified, causing the entire command to be bypassed
- 16 - Severe Error - an error of such severity occurred that not only can the command causing the error not be completed, the entire AMS command stream is flushed

IF - THEN - ELSE Structure

The statement structure used to conditionally execute commands based upon the return code values is an IF - THEN - ELSE structure:

```
IF {LASTCC | MAXCC} {operator} {numeric value}
   THEN {command} |
           DO
                 {command set}
```

```
            END
[ELSE {command} |
            DO
                  {command set}
            END]
```

In the IF statement, the return code to be tested is specified as one of LASTCC or MAXCC, where LASTCC specifies the return code set during the execution of the command just prior to the IF structure and MAXCC specifies the largest return code value set during this execution if IDCAMS. The {operator} is specified as one of the following comparison operators:

EQ or =	equal to
NE or ¬=	not equal to
GT or >	greater than
LT or <	less than
GE or >=	greater than or equal to
LE ro <=	less than or equal to

Following the THEN keyword or the optional ELSE keyword, either a single AMS command or a block of AMS commands enclosed in a DO/END pair may be coded.

Null Commands

If a THEN keyword or ELSE keyword in an IF-THEN-ELSE structure is not followed by an AMS functional command, or does not include a continuation character indicating that a functional command follows on the next line, then a null THEN or ELSE clause is assumed.

SET Command

The SET command may be used to set either the LASTCC value or the MAXCC value to a specific value. Frequently the SET command is used to reset the return code(s) to a value of 0 following an expected warning level error condition.

Defining User Catalogs

Model syntax for the command:

DEFINE **U**SER**CAT**ALOG

   (**NAME**(entryname)

   { **CYL**INDERS(primary[secondary]) |

RECORDS(primary[secondary]) |
TRAC**K**S(primary[secondary]) }

VOLUME(volser)

[**FILE**(ddname)]
[**TO**(date) | **FOR**(days)]

[**DATA** (
[**CYL**INDERS(primary[secondary]) |
RECORDS(primary[secondary]) |
TRAC**K**S(primary[secondary])]

[**INDEX** (
[**CYL**INDERS(primary[secondary]) |
RECORDS(primary[secondary]) |
TRAC**K**S(primary[secondary])]

[**CAT**ALOG(mastercatname[/password])]

Note that I have simplified the syntax diagram by excluding advanced optional parameters.

Space Allocation

Since a VSAM catalog owns the volume on which it resides, the VSAM catalog must be the first VSAM object stored on a volume. When a VSAM catalog is defined, AMS automatically defines a data space on that volume and then allocates space for the VSAM catalog from within that data space. Separate space allocation subparameters can be specified for the index and data components. If space is allocated only to the catalog as a whole, and separate SPACE subparameters are not specified for the index and data components, the entire data space (created automatically by AMS) is assigned to the catalog. In this situation, it is then necessary to use a separate AMS function command to define one or more data spaces owned by the catalog in which to create future VSAM objects. If separate index and data component space allocation subparameters are coded. the SPACE parameter for the catalog as a whole defines the size of the data space that is created, and the SPACE subparameters that are specified for the data and index components determine the portion of the data space that is assigned to the catalog. The remainder of the data space becomes available for other VSAM objects. In most cases, a cylinder of space will be adequate for catalogs.

Examples

The following jobstream will define two user catalogs - UCMVS801 to reside on 3380 volume MVS801 and UCMVS802 to reside on 3380 volume MVS802.

```
//DEFUCAT1 JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//MVS801   DD  UNIT=3380,VOL=SER=MVS801,DISP=OLD
//MVS802   DD  UNIT=3380,VOL=SER=MVS802,DISP=OLD
//SYSIN    DD  *

  DEFINE USERCATALOG (
-
               NAME (UCMVS801)
-
               VOLUME (MVS801)
-
               TRACKS (13259 0)
-
               FOR (9999) )
-
         DATA (TRACKS (15 5) )
-
         INDEX (TRACKS (15) )

  IF LASTCC = 0 THEN
-
        LISTCAT ALL CATALOG(UCMVS801)

  DEFINE USERCATALOG (
-
               NAME (UCMVS802)
-
               VOLUME (MVS802)
-
               TRACKS (15)
-
               FOR (9999) )

  IF LASTCC = 0 THEN
-
        LISTCAT ALL CATALOG(UCMVS802)

/*
//
```

The SYSOUT from this jobstream can be viewed as DEFUCAT1. The difference between the two catalogs created shows how using the SPACE parameter on the optional DATA and INDEX components can be used to define both the catalog and also leave available data space in a single operation. If you look at the catalog listing in the SYSOUT (following the AMS statements defining the user catalog UCMVS801), you can see that under MVS801's dataspace extent information 13,259 tracks have been allocated to the data space, of which only 30 have been used (to contain the user catalog). Compare this to the listing for MVS802 and you can see that only 15 tracks were allocated for the data space and all 15 have been used for the catalog. Before any VSAM objects can be created on volume MVS802, a separate data space will need to be defined. I will do that next under the Defining Data Space section.

The following jobstream will define a single user catalog - UCMVOL to reside on 3380 volume MVS803. Initially it will have all the remaining space on volume MVS803 for suballocation. In the Defining Data Space section, I will add data spaces to this catalog on two additional volumes.

```
//DEFUCAT2 JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//MVS803   DD  UNIT=3380,VOL=SER=MVS803,DISP=OLD
//SYSIN    DD  *

  DEFINE USERCATALOG (
-
               NAME (UCMVOL)
-
               VOLUME (MVS803)
-
               TRACKS (13259 0)
-
               FOR (9999) )
-
         DATA (TRACKS (15 5) )
-
         INDEX (TRACKS (15) )

  IF LASTCC = 0 THEN
-
        LISTCAT ALL CATALOG(UCMVOL)

/*
//
```

The SYSOUT from this jobstream can be viewed as DEFUCAT2. At this point there is no significant difference between this user catalog and UCMVS801 defined in the previous jobstream.

Defining Data Space

Model syntax for the command:

DEFINE **SPACE**
({**CAN**DIDATE
CYLINDERS(primary[secondary]) |
RECORDS(primary[secondary]) **REC**ORD**SIZ**E(average
maximum) |
TRAC**K**S(primary[secondary])}
VOLUMES(volser[volser...])

[**FILE**(ddname)]))

[CATALOG(catname[/password])]

A data space can be defined implicitly for a new VSAM object by coding the UNIQUE parameter in the DEFINE command for the object. This causes a new data space, of the requested size, to be defined for the sole use of that object. From a data management viewpoint, it is usually better to allocate an entire direct access storage volume as VSAM data space and suballocate space for defined objects from that space. The purpose of the DEFINE SPACE AMS command is to allocate data space and place it under the control of a user catalog.

Examples

The following jobstream will define data space using the remainder of the available space on volume MVS802, which is controlled by the user catalog UCMVS802, previously defined.

```
//DEFSPAC1 JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//MVS802   DD  DISP=OLD,UNIT=3380,VOL=SER=MVS802
//SYSIN    DD  *

  DEFINE SPACE (
-
               TRACKS(13244)
-
               VOLUMES(MVS802)
-
               FILE(MVS802)
-
               )
-
               CATALOG(UCMVS802)

  IF MAXCC = 0 THEN DO
-
        LISTCAT ALL CATALOG(UCMVS802)
-
     END

/*
//
```

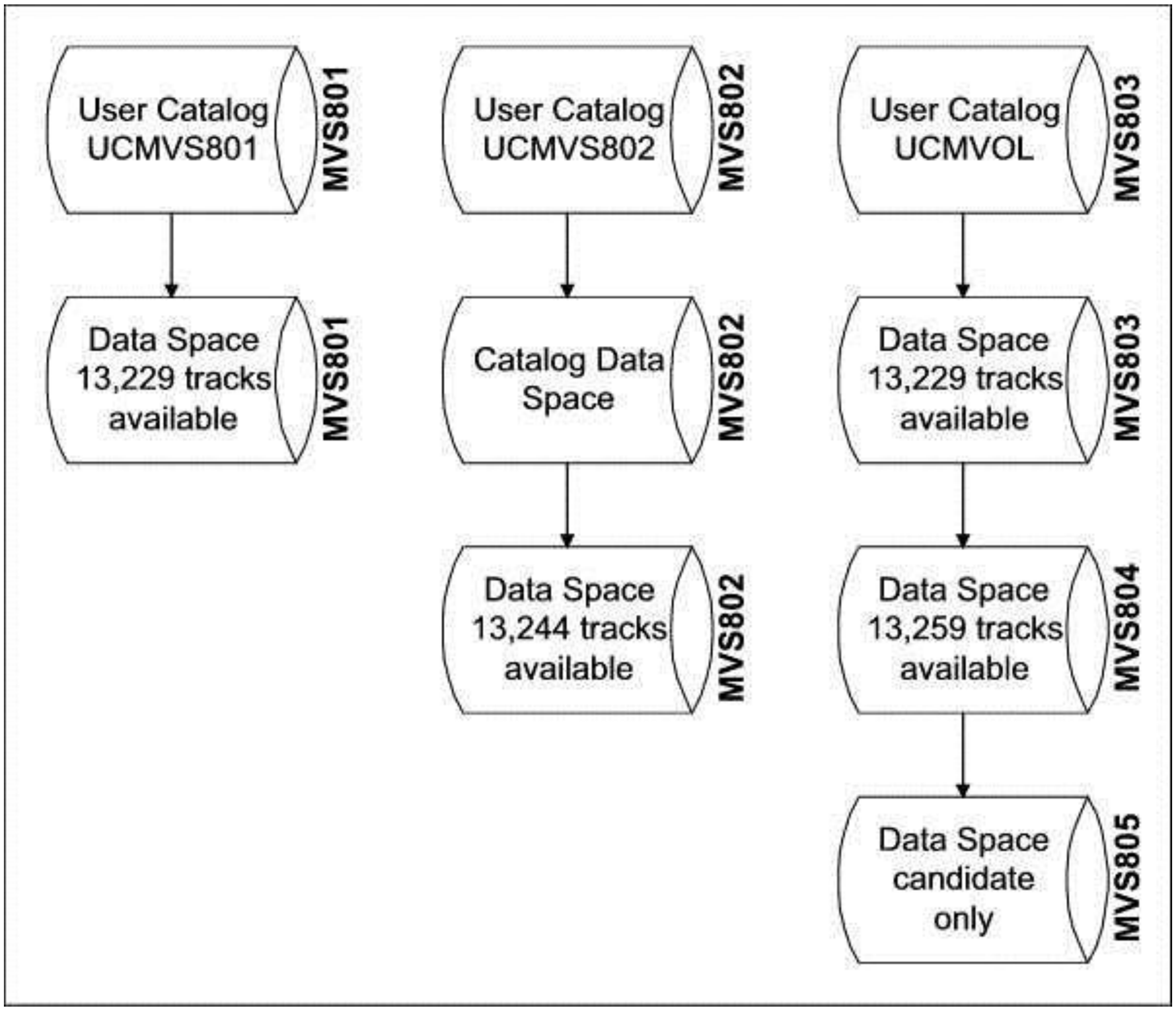
The SYSOUT from this jobstream can be viewed as DEFSPAC1. If you look at the catalog listing in the SYSOUT, you can see that under MVS802's dataspace extent information that a second DATASPACE has been added with 13,244 tracks available for allocation.

The following jobstream will define two additional data spaces for user catalog UCMVOL, one of which is only a CANDIDATE, which means no actual space is allocated, but the volume is reserved for future use by the catalog, thereby preventing any other VSAM catalog from allocating objects on it.

```
//DEFSPAC2 JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//MVS804   DD  DISP=OLD,UNIT=3380,VOL=SER=MVS804
//MVS805   DD  DISP=OLD,UNIT=3380,VOL=SER=MVS805
//SYSIN    DD  *

  DEFINE SPACE (
-
               TRACKS(13259)
-
               VOLUMES(MVS804)
-
               FILE(MVS804)
-
               )
-
               CATALOG(UCMVOL)

  DEFINE SPACE (
-
               CANDIDATE
-
               VOLUMES(MVS805)
-
               FILE(MVS805)
-
               )
-
               CATALOG(UCMVOL)

  IF MAXCC = 0 THEN DO
-
        LISTCAT ALL CATALOG(UCMVOL)
-
     END

/*
//
```

The SYSOUT from this jobstream can be viewed as DEFSPAC2. If you look at the catalog listing in the SYSOUT, you can see that information for two additional volumes has been added - MVS804 and MVS805. The DATASPACE extent information for MVS804 shows 13,259 tracks available. This is the entire space originally available on the volume since there was no space used for the catalog, which resides in the data space on volume MVS803. Although there is a volume entry for MVS805, you can see that there is no DATASPACE extent information, which is the indication that MVS805 is simply a CANDIDATE for future allocation for this catalog. A display of the Volume Table of Contents (VTOC) for volume MVS805 would show no entries made for a data space.

If you have been following the examples in sequence to this point, here is a chart showing the configuration of the user catalogs and data spaces defined:


User Catalog UCMVS801
MVS801
Data Space 13,229 tracks available
MVS801
User Catalog UCMVS802
MVS802
Catalog Data Space
MVS802
Data Space 13,244 tracks available
MVS802
User Catalog UCMVOL
MVS803
Data Space 13,229 tracks available
MVS803
Data Space 13,259 tracks available
MVS804
Data Space candidate only
MVS805

Defining an Alias

There are several ways that VSAM determines which catalog to use when a catalog search is required, either to locate a catalog entry for an existing object or to create a catalog entry one for a new object. Most AMS commands can explicitly specify which catalog is to be used by including a CATALOG parameter in the command. The inclusion of a CATALOG parameter overrides any other method for determining the catalog to use. Another way to explicitly define the catalog to be used is by including the

JOBCAT and/or STEPCAT DD statements in the JCL for the job. If neither of these methods is used to designate the catalog to use, AMS uses the high level qualifier of the object or dataset name and attempts to determine the catalog to search. If the high level qualifier matches the name of a user catalog, that user catalog is used. Otherwise, the master catalog is used.

It is possible to establish any number of *alias* entries to associate multiple high level qualifier values with a specific user catalog. The AMS command to create aliases is DEFINE ALIAS.

Model syntax for the command:

DEFINE **ALIAS**
  (**NAME**(aliasname)
   RELATE(entryname))

   [**CAT**ALOG(catname[/password])]

Example

The following jobstream will define six aliases related to the three user catalogs I have previously defined. The first five simply relate the serial number of the direct access storage device controlled by the user catalog to the catalog. For the first two user catalogs, this is a one to one relationship. However, since the third user catalog controls space on three volumes, there are three aliases defined, one for the serial number of each of the three volumes. The sixth alias is an example of how a production application might be defined as an alias to a particular user catalog.

Notice that the LISTCAT command does not specify a user catalog from which the entries are to be listed. This is because the ALIASes are defined in the master catalog, and they point to user catalogs.

```
//DEFALIAS JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  DEFINE ALIAS (NAME(MVS801)
-
                RELATE(UCMVS801) )

  DEFINE ALIAS (NAME(MVS802)
-
                RELATE(UCMVS802) )

  DEFINE ALIAS (NAME(MVS803)
-
                RELATE(UCMVOL)   )
  DEFINE ALIAS (NAME(MVS804)
-
                RELATE(UCMVOL)    )

  DEFINE ALIAS (NAME(MVS805)
-
                RELATE(UCMVOL)    )

  DEFINE ALIAS (NAME(PAYROLL)
-
                RELATE(UCMVOL)    )

  IF MAXCC = 0 THEN DO
-
        LISTCAT ALIAS
-
     END

/*
//
```

The SYSOUT from this jobstream can be viewed as DEFALIAS.

Defining a VSAM Cluster

Model syntax for the command:

DEFINE **CL**USTER
(**NAME**(entryname)
{**CYL**INDERS(primary] secondary]) |
RECORDS(primary[secondary]) |
TRACK**S**(primary[secondary])}
VOLUMES(volser[volser...])
[**BUF**FER**SP**ACE(size)]
[**C**ONTROL**I**NTERVAL**SIZ**E(size)]
[**ERAS**E | **NOERAS**E]
[**FILE**(ddname)]
[FREE**SP**ACE(CI-percent[CA-percent]|0 0)]
[**IMBED** | **NOIMBED**]
[**I**NDE**XED** | NON**I**NDE**XED** | **NUM**BERE**D**]
[**KEYS**(length offset|64 0)]
[**MODEL**(entryname[/password][catname[/password]])]
[**REC**ORD**SIZE**(average maximum)]
[**REPL**ICATE | **NOREPL**ICATE]
[**RE**USE | **NORE**USE]
[**SPEED** | **RECOVERY**]

[**TO**(date) | **FOR**(days)]
[**UNIQ**UE | <u>**SUBAL**LOCATION</u>]

[**DATA**
([**NAME**(entryname)])

[**INDEX**
([**NAME**(entryname)])

[**CAT**ALOG(catname[/password])]

Note that I have simplified the syntax diagram by excluding advanced optional parameters.

Because the DEFINE CLUSTER command is used to define all three types of VSAM clusters - Key Sequenced, Entry Sequenced, and Relative Record - the command model is large, with some parameters that are never used together and many parameters that are used only to specify advanced overrides that can best be left to default values. The three main parameter lists that may be specified are grouped by CLUSTER, DATA, and INDEX. These lists of parameters apply to the CLUSTER as a whole, the DATA component of the cluster, and the INDEX component of the cluster. Almost all parameters that may be specified for the CLUSTER can also be specified individually for the DATA and INDEX components. The INDEX component is only present when a Key Sequenced cluster is being defined. In my experience, the only parameter that is frequently supplied for the DATA and INDEX component is the NAME, otherwise AMS will generate a non-intuitive name for that component of the cluster. In a production environment, another reason for specifying individual DATA and INDEX parameters might be for a performance consideration - the INDEX and DATA components of the cluster could be split and placed on separate direct access storage volumes.

Space Allocation

One of the three space allocation subparameters must be coded to specify the size of the cluster. Space may be requested in terms of cylinders, tracks, or records. Both a primary and secondary quantity may be specified for either of the three specification units. The primary quantity is the amount of space initially allocated to the cluster. The secondary quantity is the amount of additional space allocated when the available space is completely used and an additional record is added to the cluster. The number of times that a cluster can be expanded (secondary quantities of space allocated) varies based on several factors, but may be as high as 123 times. The RECORD method of allocation space is preferred, as AMS will calculate the appropriate amount of space for the type of direct access storage device upon which the cluster is allocated.

The VOLUMES parameter specifies one or more direct access storage volumes on which space may be allocated for the cluster.

Where the space is obtained for allocation is determined by the UNIQUE / SUBALLOCATION parameter. If UNIQUE is specified, free space must exist on the VOLUMES specified; an independent data space is created on the volume and is allocated entirely to the cluster being defined. From a data management viewpoint, this is usually not a good idea. A better method is to allow AMS to suballocate the required space from an already defined VSAM data space.

Type of Cluster - Key Sequenced, Entry Sequenced, or Relative Record

The presence of the INDEXED, NONINDEXED, or NUMBERED parameters determine the type of cluster created. INDEXED specifies a Key Sequenced cluster; NONINDEXED specifies an Entry Sequenced cluster; and NUMBERED specifies a Relative Record cluster.

Record Size

The RECORDSIZE parameter specifies both the size of the logical record which can be written to the cluster and also whether the records will be fixed or variable length. If the integer values of both *average* and *maximum* are identical, the records which can be written to the cluster will be fixed length and of the size specified by the value. If the values specified differ, the records written to the cluster may be in varying length, up to the value specified for *maximum*.

Keys (INDEXED clusters only)

The KEYS parameter specifies the length and position (relative to the beginning of the record, with 0 indicating the first character) of the primary key in the records written to the cluster.

Re-Usable Clusters

The REUSE parameter allows clusters to be defined that may be reset to empty status without deleting and re-defining them. This is most often used for clusters used as work datasets.

The parameters below are optional and default values may be accepted. I have included them because they may be of utility to some Hercules/MVS 3.8 users.

Buffer Space

BUFFERSPACE is used to specify the minimum amount of buffer space required to process the dataset. The value specified affects the control interval size. AMS ordinarily chooses a control interval size large enough that two control intervals and one index record will fit in the specified amount of buffer space. Regardless of what value is coded here (or the default), the value may be overridden at execution time by JCL parameter.

Control Interval Size

In most cases, the CONTROLINTERVALSIZE parameter should be omitted. This allows AMS to choose the most efficient value for the dataset. A control interval can range from 512 to 32,768 bytes in size. If the size is between 512 and 8,192 bytes, a multiple of 512 should be specified. If it is between 8,192 and 32,768 bytes, a multiple of 2,048 should be specified. If the size is not a multiple of the appropriate value, AMS rounds the size *up* to the next appropriate multiple. If CONTROLINTERVALSIZE is specified for the INDEX component of a KSDS, the specified size must be 512, 1,024, 2,048, or 4,096.

Erase

The ERASE parameter specifies that when the cluster is deleted, the space occupied by the cluster should be physically erased by overwriting the space with binary zeros prior to freeing the space for reuse.

Free Space (INDEXED clusters only)

The FREESPACE parameter specifies a percentage of space to leave unallocated for future expansion. The percentage applies when records are initially loaded into the cluster and when control interval and control area splits occur as records are inserted between existing records. If FREESPACE is not specified, control intervals are filled as completely as possible and no space is left for addition of records in the future.

Replicate and Imbed (INDEXED clusters only)

REPLICATE specifies that VSAM should write each index record on a track as many times as it will fit. IMBED specifies that sequence set records are to be imbedded with the data in the data component of the cluster. When the sequence set is imbedded in the data component, VSAM writes each sequence set record on the first track of its associated control area. IMBED automatically implies REPLICATE. Without imbedding, the sequence set records are kept in the index component with other index records. The use of REPLICATE and IMBED may improve performance at the expense of an increase in storage requirements.

Model

The MODEL parameter is used to specify an existing cluster from which the attributes used to define the new cluster should be copied. When MODEL is used, the only additional parameter that is required is NAME. If additional parameters are specified with MODEL, they will override attributes copied from the existing cluster.

Pre-Formatting Space

The SPEED / RECOVERY parameters are used to specify whether or not VSAM should preformat the space allocated to the cluster as part of the DEFINE process. Specifying RECOVERY (the default) causes the allocated space to be filled with end-of-file markers. If the initial load of the cluster with data should fail before completion, the end-of-file markers can be used to *resume* the load from the point of failure. For large datasets, this can save recovery time, however there is a trade-off in time to write the end-of-file markers during the definition of the cluster.

Examples

The following jobstream will define a Key Sequenced cluster.

```
//DEFCLUSK JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* DELETE KSDS CLUSTER, IF IT EXISTS
*/

  DELETE MVS801.STUDENT.FILE CLUSTER PURGE

  /* DEFINE KSDS CLUSTER
*/

  DEFINE CLUSTER (
-
               NAME(MVS801.STUDENT.FILE)
-
               VOLUMES(MVS801)
-
               RECORDSIZE(80 80)
-
               RECORDS(50 10)
-
               KEYS(10 0)
-
               INDEXED )
-
         DATA (
-
               NAME(MVS801.STUDENT.FILE.DATA) )
-
         INDEX (
-
               NAME(MVS801.STUDENT.FILE.INDEX) )

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(MVS801.STUDENT)
```

```
/*
//
```

The SYSOUT from this jobstream can be viewed as DEFCLUSK. Note that by specifying the NAME parameter for both the DATA and INDEX component, it is easy to see the relationship of these two components to the CLUSTER in the catalog listing.

The following jobstream will define an Entry Sequenced cluster.

```
//DEFCLUSE JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* DELETE ESDS CLUSTER, IF IT EXISTS
*/

  DELETE PAYROLL.TIMECLOK.FILE CLUSTER PURGE

  /* DEFINE ESDS CLUSTER
*/

  DEFINE CLUSTER (
-
               NAME(PAYROLL.TIMECLOK.FILE)
-
               VOLUMES(MVS803 MVS804)
-
               RECORDSIZE(22 22)
-
               RECORDS(100 100)
-
               NONINDEXED )
-
         DATA (
-
               NAME(PAYROLL.TIMECLOK.FILE.DATA) )

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(PAYROLL.TIMECLOK)

/*
//
```

The SYSOUT from this jobstream can be viewed as DEFCLUSE. Note that since I specified two volumes for allocation of space to this dataset, the catalog listing indicates that the initial space allocation was made on volume MVS803 (VOLFLAG----PRIME indicates this is the primary extent) and that the second volume has been set as a candidate for future allocation (VOLFLAG----CANDIDATE).

The following jobstream will define a Relative Record cluster.

```
//DEFCLUSR JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* DELETE RRDS CLUSTER, IF IT EXISTS
*/

  DELETE MVS802.HRTITLE.TABLE CLUSTER PURGE

  /* DEFINE RRDS CLUSTER
*/

  DEFINE CLUSTER (
-
               NAME(MVS802.HRTITLE.TABLE)
-
               VOLUMES(MVS802)
-
               RECORDSIZE(30 30)
-
               RECORDS(100 100)
-
               NUMBERED
-
               REUSE )
-
         DATA (
-
               NAME(MVS802.HRTITLE.TABLE.DATA) )

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(MVS802.HRTITLE)

/*
//
```

The SYSOUT from this jobstream can be viewed as DEFCLUSR.

Loading Records into VSAM Clusters

Model syntax for the command:

REPRO {**IN**F**ILE**(ddname[/password]
[**ENV**IRONMENT(**DUM**MY)]) |
IN**D**ATA**S**ET(entryname[/password]

[**ENV**IRONMENT(**DUM**MY)])}

{**O**UT**FILE**(ddname[/password]) |
OUT**D**ATA**S**ET(entryname[/password])}

[**F**ROM**KEY**(key) | **F**ROM**ADDR**ESS(address) |
FROM**NUM**BER(number) | **SKIP**(count)]

[**REP**LACE | NO**REP**LACE]

[**R**E**USE** | NO**R**E**USE**]

[**TOKEY**(key) | **T**O**ADDR**ESS(address) |
TO**NUM**BER(number) | **COUNT**(count)]

Although in actual production environments the initial loading of records into VSAM clusters might require a program to be written in COBOL, PL/1, or Assembler, the AMS REPRO command can often be used for this purpose. In later sections I will show how the REPRO command is also used for other purposes, but here the goal is the initial loading of VSAM clusters from source data contained in a another dataset.

The only two required parameters are those which specify the input dataset (the source of the records to be loaded) and the output cluster, which is to be loaded. The input may be specified by either the INFILE parameter or the INDATASET parameter. When INFILE is used, the records are read from the DD name specified, and a DD statement must be supplied in the JCL matching the specified name. When INDATASET is used, the source of the input is located by searching the VSAM catalog for a matching object.

The ENVIRONMENT subparameter is used to specify, in the case of ISAM input objects, that DUMMY records are to be included along with active records.

The VSAM cluster to receive the records read from the input dataset is specified by either OUTFILE or OUTDATASET. OUTFILE designates that the target cluster is specified by the inclusion of a DD statement in the JCL, while OUTDATASET designates that the cluster is to be located through the VSAM catalog.

Bypassing Input Records

FROMKEY may be included to specify either the full or generic key value that defines the starting point of the copy operation. FROMKEY is only applicable when the input dataset is a KSDS VSAM cluster or an ISAM dataset.

FROMADDRESS may be included to specify the RBA value that defines the starting point of the copy operation. FROMADDRESS is only applicable when the input dataset is a KSDS or ESDS VSAM cluster.

FROMNUMBER may be included to specify the relative record number that defines the starting point of the copy operation. FROMNUMBER is only applicable when the input dataset is a RRDS VSAM cluster.

SKIP may be included to bypass the specified number of records from the input dataset before starting the copy operation.

Restarting an Aborted Partial Load

REPLACE may be specified to cause existing records in the output cluster to be replaced when a duplicate record is read from the input dataset. REPLACE is applicable for KSDS and RRDS clusters. REPLACE may also be used to merge input from subsequent REPRO operations with different input datasets to the same target dataset, where the possibility of duplicate records. If an attempt is made to add a duplicate record and REPLACE is not specified, an error message and the input record is discarded; on the fourth occurrence of a duplicate record, the REPRO operation is terminated and the remainder of the input file is not processed.

Resetting the Output Cluster to Empty

The REUSE parameter may be specified to cause the output cluster to be reset to empty status before loading commences. In order for the REUSE parameter to be allowable, the output cluster must have been defined with the REUSE attribute.

Ending the Load Operation Before End of Input

TOKEY may be included to specify either the full or generic key value that defines the ending point of the copy operation. TOKEY is only applicable when the output cluster is a KSDS VSAM cluster.

TOADDRESS may be included to specify the RBA value that defines the ending point of the copy operation.. TOADDRESS is only applicable when the output dataset is a KSDS or ESDS VSAM cluster.

TONUMBER may be included to specify the relative record number that defines the ending point of the copy operation. TONUMBER is only applicable when the output dataset is a RRDS VSAM cluster.

COUNT may be included to terminate the copy operation after the specified number of records have been copied to the output cluster.

Examples

The following jobstream will load Relative Record cluster MVS802.HRTITLE.TABLE using data from an instream file, which is first processed by the IEBDG utility to trim the records to the correct record length.

```
//REPRO01  JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A

   ------[ IEBDG step and instream data omitted ]------

//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//SEQFILE  DD  DSN=*.IEBDG.SYSUT2,DISP=(OLD,DELETE)
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  REPRO INFILE(SEQFILE)
-
        OUTDATASET(MVS802.HRTITLE.TABLE)
-
        REUSE

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(MVS802.HRTITLE)

/*
//
```

The SYSOUT from this jobstream can be viewed as REPRO01. Because the cluster was defined with the REUSE attribute, specifying the REUSE parameter on the REPRO command resets the status of the output cluster to empty before any records are copied. If more records later need to be added to this table, they simply need to be added to the instream dataset and the job resubmitted to reload the cluster, replacing the existing records with the records from the instream dataset.

The following jobstream will load Key Sequenced cluster MVS801.STUDENT.FILE using data from an instream dataset.

```
//REPRO02  JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//SEQFILE  DD  *

   ------[ instream data omitted ]------

/*
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  REPRO INFILE(SEQFILE)
-
        OUTDATASET(MVS801.STUDENT.FILE)
-
        REPLACE

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(MVS801.STUDENT)
```

```
/*
//
```

The SYSOUT from this jobstream can be viewed as REPRO02. This cluster was not defined as REUSEable, however, specifying the REPLACE parameter on the REPRO command causes records in the output cluster with key values identical to records copied from the input dataset to be replaced. I submitted this job twice to illustrate - note that under the DATA component in the catalog listing the total number of records is 97, the count of records processed. The number of records retrieved and updated is also 97, which indicates that the second time the job was submitted, all of the records read from the input dataset had a matching record already existing in the output cluster.

Using REPRO to Backup Clusters

The REPRO command can also be used to make backup copies of the contents of VSAM clusters.

Examples

The following jobstream will create a backup copy of the contents of Entry Sequenced cluster PAYROLL.TIMECLOK.FILE on tape.

```
//REPRO03  JOB 'JAY MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//*
//************************************************************
//* CREATE A SEQUENTIAL BACKUP OF AN ESDS
//************************************************************
//*
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//TIMECLOK DD  DSN=PAYROLL.TIMECLOK.FILE,DISP=SHR
//BACKUP   DD  DSN=BACKUP.TIMECLOK,UNIT=TAPE,DISP=(NEW,KEEP),
//             VOL=SER=B00001,
//             DCB=(RECFM=FB,LRECL=22,BLKSIZE=2200)
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  REPRO INFILE(TIMECLOK)                                      -
        OUTFILE(BACKUP)

/*
//
```

The SYSOUT from this jobstream can be viewed as REPRO03.

The following jobstream will create a backup copy of the contents of KS cluster MVS801.STUDENT.FILE in another KS cluster, MVS802.STUDENT.BACKUP.

```
//REPRO04  JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* DELETE BACKUP CLUSTER, IF IT EXISTS
*/

  DELETE MVS802.STUDENT.BACKUP CLUSTER PURGE

  /* DEFINE BACKUP KSDS CLUSTER
*/

  DEFINE CLUSTER (
-
               NAME(MVS802.STUDENT.BACKUP)
-
               VOLUMES(MVS802)
-
               MODEL(MVS801.STUDENT.FILE) )
-
         DATA (
-
               NAME(MVS802.STUDENT.BACKUP.DATA) )
-
         INDEX (
-
               NAME(MVS802.STUDENT.BACKUP.INDEX) )


  IF LASTCC = 0 THEN
-
        REPRO  INDATASET(MVS801.STUDENT.FILE)
-
               OUTDATASET(MVS802.STUDENT.BACKUP)

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(MVS802.STUDENT)

/*
//
```

The SYSOUT from this jobstream can be viewed as REPRO04. The target of the REPRO is first deleted and then defined, using the MODEL parameter to copy the attributes of the source KSDS. If this were an actual backup strategy, it would probably be better to have the backup cluster defined once, at the same time as the cluster to be backed up, but with the addition of the REUSE attribute. Then the REPRO command could use the REUSE parameter to reset the backup dataset to empty prior to the copy operation in order to save the time to delete and re-define it each run.

Restoring from Backup

Restoring the contents of a VSAM cluster from a backup created in the examples above is identical to the process used to load VSAM clusters with REPRO. Unless the cluster is defined with the REUSE attribute, it is necessary to first delete and then redefine the cluster in order to remove the existing records that will be replaced with those from the backup. Then execute a jobstream containing the REPRO command using the backup copy as the source for the input.

Exporting VSAM Clusters

The EXPORT command is used to create a portable copy of a VSAM cluster. When the REPRO command is used to copy a cluster, only the data records are written to the output dataset. Before the records can be restored, the cluster must be recreated by executing an AMS DEFINE command. EXPORT copies the information required to completely reconstruct the cluster without the necessity of executing a separate DEFINE. The portable copy of the exported cluster must always be contained in a non-VSAM, sequential dataset.

Model syntax for the command:

EXPORT entryname[/password]
{**O**UT**FILE**(ddname) | **O**UT**D**ATA**S**ET(entryname)}
[**ERAS**E | **NOERAS**E]
[**I**N**FILE**(ddname)]
[**INH**IBIT**S**OURCE | **NOINH**IBIT**S**OURCE]
[**INH**IBIT**T**ARGET | **NOINH**IBIT**T**ARGET]
[P**U**R**G**E | **NOP**U**RG**E]
[**TEMP**ORARY | **PERM**ANENT]

Note that there is no CATALOG parameter for the EXPORT command. Unless the object's entry is in the master catalog or the first qualifier of the object's name is an alias of the user catalog containing the object, you must include either a JOBCAT or STEPCAT DD statement in the JCL.

When EXPORT is executed with the PERMANENT option (the default), if the portable copy of the cluster is successfully created, AMS will attempt to delete the original object. By default, the attributes of the cluster which affect a delete function that are in effect at the time of the EXPORT will control the delete processing. If the ERASE attribute was specified for the cluster, the cluster will be overwritten with binary zeros before it is deleted. The ERASE / NOERASE option may be specified on the EXPORT command to override the attribute specified when the cluster was defined. If a retention period was specified for the cluster and has not been reached, the delete will not take place. The PURGE / NOPURGE option may be specified on the EXPORT command to override the retention status of the source cluster.

INHIBITSOURCE may be specified to mark the source cluster as "read only" after the successful completion of the EXPORT. Likewise, INHIBITTARGET may be specified to mark any copy of the cluster created from the portable copy as "read only".

Example

The following jobstream will create a temporary portable copy of the Relative Record cluster MVS802.HRTITLE.TABLE on tape. Note that since I have specified the backup tape disposition as CATLG and the high level qualifier of the backup dataset name is MVS802, the tape dataset will be catalogued in the same user catalog as the VSAM cluster from which it is copied.

```
//EXPORT1  JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//BACKUP   DD  DSN=MVS802.HRTITLE.BACKUP,
//             UNIT=TAPE,DISP=(NEW,CATLG)
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  EXPORT MVS802.HRTITLE.TABLE
-
         OUTFILE(BACKUP)
-
         TEMPORARY

/*
//
```

The SYSOUT from this jobstream can be viewed as EXPORT1. Because TEMPORARY was specified on the EXPORT command, the source cluster remains after the portable copy is created.

Restoring Exported Clusters

A sequential dataset created by the EXPORT command may only be processed as input to the IMPORT command, which will result in an equivalent copy of the original source cluster. AMS first searches the target catalog for an entry with the same name as the cluster being imported. If an entry does not exist, AMS uses the catalog information included in the portable copy to define a new catalog entry, and then it reloads the object. If a catalog entry does exist, AMS verifies that the entry has been marked as exported by a previous EXPORT command with the TEMPORARY option. If this is the case, AMS assumes that the cluster is being restored from the previously exported copy; AMS deletes the old object, defines a new catalog entry of it using the catalog information in the exported copy, and then reloads the object. If a catalog entry exists, but it is not marked as having been previously exported, the IMPORT command fails.

Model syntax for the command:

IMPORT {**I**N**FILE**(ddname) | **I**N**D**ATA**S**ET(entryname)}
{**O**UT**FILE**(ddname[/password]) |
OUT**D**ATA**S**ET(entryname[/password])}
[**ERAS**E | **N**O**ERAS**E]
[INTO**EMPTY**]
[**OBJ**ECTS((entryname
[**FILE**(ddname)]
[**NEWNAM**E(newname)]
[**VOL**UMES(volser[volser...])])
[(entryname...)...])]
[**P**U**RG**E | **N**O**P**U**RG**E]

[**CAT**ALOG(catname[/password])]

Note that I have simplified the syntax diagram by excluding advanced optional parameters.

If the non-VSAM dataset containing the portable copy is cataloged, the INDATASET parameter may be used to access the input dataset.

Example

The following jobstream will restore the Relative Record cluster MVS802.HRTITLE.TABLE from the portable copy which exists on tape.

```
//IMPORT1  JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//BACKUP   DD  DSN=MVS802.HRTITLE.BACKUP,
//             UNIT=TAPE,DISP=(OLD,KEEP)
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  IMPORT INFILE(BACKUP)
-
         OUTDATASET(MVS802.HRTITLE.TABLE)
-
         PURGE

/*
//
```

The SYSOUT from this jobstream can be viewed as IMPORT1.

In addition to being used for making backup copies, the EXPORT and IMPORT commands can also be used to move VSAM clusters from one computer system to another. Several of the parameter options on the IMPORT command are designed specifically for this purpose.

The OBJECTS parameter specifies the original name of the object(s) being imported and allows a new name and alternate target volumes to be specified, which override those contained in the portable copy. Other attributes of the restored cluster may be changed with the ALTER command. A few attributes, including space allocation and some of the attributes which affect performance can not be altered. If it is necessary to change those attributes, it is necessary to define a new empty cluster with the desired attributes and specify the INTOEMPTY parameter. The INTOEMPTY parameter specifies that the portable copy is being imported into an already existing empty cluster. Remember the MODEL parameter of the DEFINE command can be used to easily create a cluster that is based on another cluster which closely approximates the attributes desired for the target cluster, and then additional parameters can be specified to exactly tailor the DEFINE command.

Alternate Indexes

If you are reading this page in sequence, I have not discussed alternate indexes yet, but there are implications for using EXPORT and IMPORT that affect alternate indexes that need to be mentioned.

The EXPORT and IMPORT commands can be used to back up and restore alternate indexes as well as clusters. When the PERMANENT option is used to export an alternate index, all paths associated with the alternate index are deleted; and when a cluster that has paths and alternate indexes associated with it is exported using the PERMANENT option, all alternate index and path entries are automatically deleted. This means that alternate indexes associated with a base cluster should be exported before the base cluster itself is exported. And when these objects are then imported, the base cluster should be imported first, followed by all of its alternate indexes. Backing up and restoring in this sequence ensures that all catalog entries for the cluster, its alternate indexes, and its paths are backed up and restored properly.

Backing Up and Restoring Catalogs

The REPRO command can be used to copy the contents of user catalogs as a means of creating a backup copy that may later be restored. When using REPRO to restore from a backup copy of a user catalog, the output catalog must be empty.

Restoring a user catalog from a backup is not an operation for the faint hearted. There are many pieces of related information that must mesh in order for VSAM catalogs to operate properly. It only takes a small error to render the entire structure unusable. Unfortunately, with MVS 3.8j we don't have access to a better method of backing up and restoring catalog information.

Examples

The following jobstream will create a backup of the contents of user catalog UCMVS802 on tape.

```
//REPROUC1 JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//STEPCAT  DD  DSN=UCMVS802,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//BACKUP   DD  UNIT=TAPE,DISP=(NEW,KEEP),
//             DSN=BACKUP.UCMVS802,
//             DCB=BLKSIZE=31744
//SYSIN    DD  *

  REPRO INDATASET(UCMVS802)
-
        OUTFILE(BACKUP)

/*
//
```

The SYSOUT from this jobstream can be viewed as REPROUC1.

The following jobstream will restore user catalog UCMVS802 from the backup copy created in the previous jobstream. Because the target of the REPRO must be empty, it is necessary to first delete and redefine the user catalog.

```
//REPROUC2 JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS1  EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  DELETE (UCMVS802)
-
         USERCATALOG FORCE PURGE

  IF LASTCC = 0 THEN
-
        DEFINE USERCATALOG (
-
                     NAME (UCMVS802)
-
                     VOLUME (MVS802)
-
                     TRACKS (15)
-
                     FOR (9999) )

/*
//IDCAMS2  EXEC PGM=IDCAMS,REGION=4096K,COND=(0,NE)
//STEPCAT  DD  DSN=UCMVS802,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//BACKUP   DD  UNIT=(TAPE,,DEFER),DISP=(OLD,KEEP),
//             DSN=BACKUP.UCMVS802,VOL=SER=B00002
//SYSIN    DD  *

  REPRO INFILE(BACKUP)
```

```
-
           OUTDATASET(UCMVS802)

/*
//
```

The SYSOUT from this jobstream can be viewed as REPROUC2. The messages issued for each of the objects restored:

IDC1575I ONLY BACKUP DEFINES <object name>

are issued because the information used to create the catalog entry for the object is obtained solely from the backup tape.

Deleting VSAM and non-VSAM Objects

The DELETE command removes the entry for the specified object(s) from the catalog and optionally removes the object, thereby freeing up the space occupied by the object (in the case of datasets residing on direct access storage).

Model syntax for the command:

DELETE (entryname[/password][entryname[/password] ...])
   [**ALIAS** |
   ALTERNATE**I**NDE**X** |
   CLUSTER |
   GENERATION**D**ATA**G**ROUP |
   NON**VSAM** |
   PAGE**SP**ACE |
   PATH |
   SPACE |
   USER**CAT**ALOG

   [**ERAS**E | NO**ERAS**E]
   [**FILE**(ddname)]
   [**F**O**R**CE | **NOF**O**R**CE]
   [**P**U**RG**E | **NOP**U**RG**E]
   [**SCR**ATCH | **NOSCR**ATCH]

   [**CAT**ALOG(catname[/password])]

From the model, it is obvious that a variety of VSAM and non-VSAM objects may be deleted with the DELETE command. The DELETE command deletes all objects associated with the entry name specified.

By default, objects with a retention period that has not expired will not be deleted. This behavior can be overridden by the inclusion of the PURGE option.

The ERASE / NOERASE option may be specified to override the ERASE attributed specified for the object in the catalog.

The FORCE option may be specified to cause the deletion of specific objects (SPACE, USERCATALOG, GENERATIONDATAGROUP) even though they may be non-empty.

The SCRATCH option may be specified to cause the associated entry for the object to be removed from the Volume Table of Contents. This is most often applicable to non-VSAM datasets.

The CATALOG option may not be specified for DELETE commands used to delete user catalogs. User catalog entries may only be deleted from the master catalog. **Note: It is possible with MVS 3.8j to catalog a user catalog entry in another user catalog, but if you do, you cannot delete it.**

Example

The following jobstream will delete a non-VSAM dataset, a user catalog, and a user catalog alias. You may also have noticed the DELETE command used in several earlier jobstreams, which can provide additional examples.

```
//DELETE1  JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* DELETE CATALOGUED NON-VSAM DATASET
*/

  DELETE (MVS501.REGISTER.DATA)
-
         NONVSAM SCRATCH PURGE

  /* DELETE ALIAS TO USER CATALOG
*/

  DELETE (MVS501)
-
         ALIAS

  /* DELETE USER CATALOG
*/

  DELETE (UCMVS501)
-
         USERCATALOG FORCE PURGE
```

```
/*
//
```

The SYSOUT from this jobstream can be viewed as DELETE1. The entry for the non-VSAM dataset is removed from both the catalog (equivalent to the UNCATLG function in the IEHPROGM utility) and from the VTOC on volume MVS501 (equivalent to the SCRATCH function in the IEHPROGM utility) because the SCRATCH option was included.

Listing Catalog Entries

The LISTCAT command is provided to list the information from catalog entries. If you have been reading this page in sequence, you have already seen output from the LISTCAT command in the SYSOUT listings from my examples of many of the other AMS commands. It is almost always a good idea to list the catalog entries for objects immediately following their creation to visually verify that all of the options you intended to specify were entered correctly and had the desired effect on the entry created. It is also frequently necessary to list fields from the catalog entry for a VSAM object to diagnose problems.

Model syntax for the command:

LISTCAT
[**ALIAS**] |
[**A**LTERNATE**I**NDE**X**] |
[**CL**USTER] |
[**DATA**] |
[**G**ENERATION**D**ATA**G**ROUP] |
[**I**NDE**X**] |
[**N**ON**VSAM**] |
[**P**A**G**E**SP**ACE] |
[**PATH**] |
[**SPACE**] |
[**U**SER**CAT**ALOG]

[**CREAT**ION(days)]
[**ENT**RIES(entry name[/password] [entryname[/password]...]) |
LEVEL(level)]
[**EXPIR**ATION(days)]
[<u>**NAME**</u> | **HIST**ORY | **VOL**UME | **ALLOC**ATION | **ALL**]
[**O**UT**FILE**(ddname)]

[**CAT**ALOG(catname[/password])]

All parameters of the LISTCAT command are optional.

The first group of parameters (beginning with ALIAS and concluding with USERCATALOG) are positional parameters that specify which types of catalog entries are to be listed. One or more of these parameters may be specified. If none are specified, the listing will include all entries from the catalog regardless of type.

CREATION specifies a number of days; entries are listed only if they were created the specified number of days ago or earlier.

EXPIRATION also specifies a number of days; entries are listed only if they will expire in the specified number of days or earlier.

The parameters NAME, HISTORY, VOLUME, ALLOCATION, and ALL specify the type of information to list for catalog entries. NAME (the default) specifies that only the name and entry type should be listed. HISTORY specifies that the name, entry type, ownerid, creation date, and expiration date should be listed. VOLUME specifies that all information listed by the HISTORY parameter, plus the volume serial numbers and device type should be listed. ALLOCATION specifies that all information listed by the VOLUME parameter, plus the detail information about space allocation should be listed. ALL specifies that all information from the catalog entry should be listed.

OUTFILE may be used to specify that the output from the LISTCAT command should be written to a DD name other than SYSPRINT.

Selecting Objects to List

Either the ENTRIES or LEVEL parameter is used to select objects in the catalog for which the entries will be listed. The names of one or more objects can be included as subparameters of the ENTRIES parameter to select specific objects. It is also possible to use generic names to select groups of entries based upon the structure of their names. A generic name uses an asterisk (*) instead of a node name at one or more levels to refer to objects that have any node names at that level. The LEVEL provides an alternative for specifying a generic name. When one or more initial qualifiers are specified in a LEVEL parameter, the parameter refers to all objects whose names begin with the specified qualifiers, no matter how long their names are.

Example

The following jobstream illustrates several options of the LISTCAT command.

```
//LISTCAT1 JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* LIST ALIAS ENTRIES FROM MASTER CATALOG
*/
```

```
  LISTCAT ALIAS
-
          NAME

  /* LIST NAMES OF ALL ENTRIES FROM CATALOG UCMVOL
*/

  LISTCAT NAME
-
          CAT(UCMVOL)

  /* LIST ENTIRE ENTRY FOR NON-VSAM ENTRIES IN UCMVS802
*/

  LISTCAT NONVSAM
-
          ALL
-
          CAT(UCMVS802)

  /* LIST ALLOCATION INFO FOR DATA IN UCMVS801
*/

  LISTCAT DATA
-
          ALLOCATION
-
          CAT(UCMVS801)

/*
//
```

The SYSOUT from this jobstream can be viewed as LISTCAT1.

Printing the Contents of Objects

The PRINT command is used to print the contents of both VSAM and non-VSAM datasets.

Model syntax for the command:

PRINT {**I**N**FILE**(ddname[/password]) |
          IN**D**ATA**S**ET(entryname[/password])}
   [**CHAR**ACTER | **<u>DUMP</u>** | **HEX**]
   [**F**ROM**KEY**(key) | **F**ROM**ADDR**ESS(address) |
     FROM**NUM**BER(number) | **SKIP**(number)]
   [**O**UT**FILE**(ddname)]
   [**TOKEY**(key) | **TOADDR**ESS(address) |
     TONUMBER(number) | **COUNT**(number)]

The only required parameter is either INFILE or INDATASET to specify the dataset from which records are to be printed. When INFILE is used, the records are read from the DD name specified, and a DD statement must be supplied in the JCL matching the specified name. When INDATASET is used, the source of the input is located by searching the VSAM catalog for a matching object. There is no CATALOG option for the PRINT command, so if INDATASET is used and there is no alias defined for the catalog in which the object can be located, it will be necessary to use either a JOBCAT or STEPCAT DD statement for the proper catalog.

Bypassing Input Records

FROMKEY may be included to specify either the full or generic key value that defines the starting point of the print operation. FROMKEY is only applicable when the input dataset is a KSDS VSAM cluster or an ISAM dataset.

FROMADDRESS may be included to specify the RBA value that defines the starting point of the print operation. FROMADDRESS is only applicable when the input dataset is a KSDS or ESDS VSAM cluster.

FROMNUMBER may be included to specify the relative record number that defines the starting point of the print operation. FROMNUMBER is only applicable when the input dataset is a RRDS VSAM cluster.

SKIP may be included to bypass the specified number of records from the input dataset before printing is to begin.

Ending the Print Operation Before End of Input

TOKEY may be included to specify either the full or generic key value that defines the ending point of the print operation. TOKEY is only applicable when the output cluster is a KSDS VSAM cluster.

TOADDRESS may be included to specify the RBA value that defines the ending point of the print operation.. TOADDRESS is only applicable when the output dataset is a KSDS or ESDS VSAM cluster.

TONUMBER may be included to specify the relative record number that defines the ending point of the print operation. TONUMBER is only applicable when the output dataset is a RRDS VSAM cluster.

COUNT may be included to terminate the print operation after the specified number of records have been printed.

Output Format

CHARACTER, DUMP (the default), or HEX may be specified to control the format of the output. CHARACTER specifies that each byte in the logical record is to be printed as a character. DUMP specifies that each byte in the logical record is to be printed in both hexadecimal and character format. HEX specifies that each byte in the logical record is to be printed as two hexadecimal digits.

Examples

The following jobstream illustrates the use of the PRINT command with three datasets, one non-VSAM and two VSAM.

```
//PRINT1   JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//HRTABLE  DD   DSN=MVS501.SERVICES.DATA,DISP=OLD,
//              UNIT=3350,VOL=SER=MVS501
//TIMECLOK DD   DSN=PAYROLL.TIMECLOK.FILE,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *

  /* PRINT NON-VSAM DATASET
*/

  PRINT INFILE(HRTABLE)
-
        CHAR COUNT(10)

  /* PRINT ESDS DATASET
*/

  PRINT INFILE(TIMECLOK)
-
        HEX SKIP(500) COUNT(25)

  /* PRINT KSDS DATASET
*/

  PRINT INDATASET(MVS801.STUDENT.FILE)
-
          FROMKEY(00003) TOKEY(00004)
-
          DUMP

/*
//
```

The SYSOUT from this jobstream can be viewed as PRINT1.

Changing Object Attributes

The ALTER command can be used to change most of the attributes that are specified at the time a VSAM object is DEFINEd.

Model syntax for the command:

ALTER entryname[/password]
[**A**DD**VOL**UMES(volser[volser...])]
[**BUF**FER**SP**ACE(size)]
[**EMP**TY | **N**O**EMP**TY]
[**ERAS**E | **N**O**ERAS**E]
[**F**REE**SP**A**C**E(CI-percent[CA-percent])]
[**KEYS**(length offset)]
[**NEWN**A**M**E(newname)]
[**REC**ORD**SIZ**E(average maximum)]
[**R**EMOVE**VOL**UMES(volser[volser...])]
[**SCR**ATCH | **N**O**SCR**ATCH]
[**TO**(date) | **FOR**(days)]
[**UPG**RADE | **N**O**UPG**RADE]

[**CAT**ALOG(catname[/password])]

Note that I have simplified the syntax diagram by excluding advanced optional parameters.

For the majority of the parameters, the meaning is identical as when the parameter is used in the corresponding DEFINE command, so I will not be repeat any of that detail here.

Alternate Indexes

The Alternate Index capabilities of VSAM allow the creation of one or more secondary index structures for a VSAM object based upon data in the records of the object other than the primary key. This capability allows for a Key Sequenced dataset to be accessed in a different order other than the primary key and also allows the creation of an index for a non-indexed object, such as an Entry Sequenced cluster.

There are three steps to creating an alternate index:

1. Use the DEFINE command to create the alternate index
2. Use the BLDINDEX command to build the keys for the alternate index
3. Use the DEFINE command to create a PATH relating the alternate index to the base cluster

The Alternate Index cluster is a Key Sequenced cluster that is used to hold the alternate key values along with their associated pointers to the records in the base cluster.

Model syntax for the DEFINE ALTERNATEINDEX command:

DEFINE **A**LTERNATE**I**NDE**X**
   (**NAME**(entryname)
   RELATE(entryname[/password])
   {**CYL**INDERS(primary[secondary]) | **REC**ORDS(primary[
secondary]) |
     TRAC**K**S(primary[secondary])}
   VOLUMES(volser[volser...])
   [**BUF**FER**SP**ACE(size)]
   [**C**ONTROL**I**NTERVAL**SIZ**E(size)]
   [**ERAS**E | NO**ERAS**E]
   [**FILE**(ddname)]
   [**F**REE**SP**A**C**E(CI-percent[CA-percent]|0 0)
   [**IMB**ED | N**O**IMB**E**D]
   [**KEYS**(length offset|64 0)]
   [**MODEL**(entryname[/password][catname[/password
   [**REC**ORD**S**I**Z**E(average maximum|4086 32600)]
   [**REPL**ICATE | **NOREPL**ICATE]
   [**REUS**E | NO**REUS**E]
   [**SPEED** | **REC**OV**ERY]**
   [**TO**(date) | **FOR**(days)]
   [**UNIQ**UE | **SUBAL**LOCATION]
   [**U**NI**Q**UE**K**EY | NON**UNIQ**UE**K**EY]
   [**UPG**RADE | **N**O**UPG**RADE])

  [**DATA**
   ([**NAME**(entryname)])

  [**INDEX**
   ([**NAME**(entryname)])

  [**CAT**ALOG(catname[/password])]

Note that I have simplified the syntax diagram by excluding advanced optional parameters.

Many of the entries are identical to the DEFINE command for a Key Sequenced cluster. I will not repeat the detail discussion of those here.

RELATE names the existing base cluster for which the alternate index is to be defined. The cluster named may be either a KSDS or ESDS cluster. The cluster may not be defined with the REUSE attribute, may not be a user catalog, and may not be another alternate index.

UNIQUEKEY specifies that there is only one record in the base cluster corresponding to each alternate key in the alternate index. NONUNIQUEKEY (the default) specifies that each record in the base cluster may have multiple alternate keys in the alternate index.

UPGRADE specifies that the alternate index will automatically be kept up to date when records are modified in the base cluster. NOUPGRADE specifies that the alternate index will not be kept up to date. There is a performance penalty when UPGRADE is specified, but there may be little utility in having an alternate index that is not kept "in sync" with the base cluster.

The RECORDSIZE refers not to the size of the base cluster record, but is computed based upon the size of the keys in both the base cluster and the alternate index and whether the keys are unique not nonunique. If the keys are unique, the records in the alternate index are fixed length, and the length is the value computed as

- 5 bytes of control information
- the length of the alternate key
- the length of the primary key (for KSDS) or the length of an RBA (for ESDS), which is 4 bytes

As for a fixed length KSDS, this length is specified for both the average and maximum value in the RECORDSIZE parameter.

If the keys are nonunique, the records in the alternate index will be variable in length, and the average length is computed as:

- 5 bytes of control information
- the length of the alternate key
- the length of the primary key (for KSDS) or the length of an RBA (for ESDS), which is 4 bytes **multiplied by** the expected average number of nonunique keys

The maximum length is computed as:

- 5 bytes of control information
- the length of the alternate key
- the length of the primary key (for KSDS) or the length of an RBA (for ESDS), which is 4 bytes **multiplied by** the maximum number of nonunique keys

The contents of the 5 bytes of control information stored in the alternate index record indicate the type of the base cluster the alternate index points to and the length of the keys:

- the first byte indicates the type of base cluster - x'01' indicates KSDS; x'00' indicates ESDS
- the second byte indicates the length of the base cluster pointers in the alternate index record - x'primary key length' if the base

cluster is KSDS **or** x'04' if the base cluster is ESDS
- the third and fourth bytes are a halfword value indicating the number of occurrences of the base cluster pointers within the alternate index record; e.g. will contain x'0001' for a unique alternate key
- the fifth byte indicates the length of the alternate key

Examples

The following jobstream illustrates the use of DEFINE to create two alternate indexes, one over an existing Key Sequenced Data Set and one over an existing Entry Sequenced Data Set.

```
//DEFAIX1  JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//TIMECLOK DD  DSN=PAYROLL.TIMECLOK.FILE,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* DELETE ALTERNATE INDEX, IF IT EXISTS
*/

  DELETE MVS801.STUDENT.AIX ALTERNATEINDEX PURGE

  /* DEFINE ALTERNATE INDEX WITH UNIQUE KEYS -> KSDS
*/

  DEFINE ALTERNATEINDEX (
-
               NAME(MVS801.STUDENT.AIX)
-
               RELATE(MVS801.STUDENT.FILE)
-
               VOLUMES(MVS801)
-
               RECORDSIZE(37 37)
-
               TRACKS(10 5)
-
               KEYS(22 10)
-
               UNIQUEKEY
-
               UPGRADE )
-
         DATA (
-
```

```
               NAME(MVS801.STUDENT.AIX.DATA) )
-
         INDEX (
-
               NAME(MVS801.STUDENT.AIX.INDEX) )

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(MVS801.STUDENT)

  /* DELETE ALTERNATE INDEX, IF IT EXISTS
*/

  DELETE PAYROLL.TIMECLOK.AIX ALTERNATEINDEX PURGE

  /* DEFINE ALTERNATE INDEX WITH NONUNIQUE KEYS -> ESDS
*/

  DEFINE ALTERNATEINDEX (
-
               NAME(PAYROLL.TIMECLOK.AIX)
-
               RELATE(PAYROLL.TIMECLOK.FILE)
-
               FILE(TIMECLOK)
-
               VOLUMES(MVS803 MVS804)
-
               RECORDSIZE(18 900)
-
               TRACKS(15 15)
-
               KEYS(9 12)
-
               NOUPGRADE )
-
         DATA (
-
               NAME(PAYROLL.TIMECLOX.AIX.DATA) )
-
         INDEX (
-
               NAME(PAYROLL.TIMECLOX.AIX.INDEX) )

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(PAYROLL.TIMECLOX)

/*
//
```

The SYSOUT from this jobstream can be viewed as DEFAIX1. At this point, all that has been created is a cluster to contain the alternate keys. In the next step, the keys will be built from the base cluster with the BLDINDEX command.

Model syntax for the BLDINDEX command:

BLD**I**NDE**X** {INFILE(ddname[/password]) |
IN**D**ATA**S**ET(entryname[/password])}
   {**O**UT**FILE**(ddname[/password] [ddname[/password]...]) |
        OUT**D**ATA**S**ET(entryname[/password] [
entryname[/password]...])}
   [**W**ORK**FILES**(ddname ddname)]

   [**CAT**ALOG(catname[/password])]

Note that I have simplified the syntax diagram by excluding advanced optional parameters.

AMS reads the base cluster sequentially and constructs alternate key/pointer pairs from each data record in the base cluster. If the base cluster is a KSDS, each pointer consists of a prime key value; if the base cluster is an ESDS, each pointer consists of an RBA pointer.

AMS sorts the alternate key/pointer pairs to load records into ascending key sequence. For a large base cluster, the sort may have to be carried out using work files, which will require DD names in the JCL.

The sorted alternate key/pointer pairs are used to load records into the data component of the alternate index. Control information is added to each alternate key/pointer pair that tells VSAM the length of the alternate key, describes whether the pointers are prime keys or RBA's, indicates whether the keys are unique or nonunique, and gives the length of each prime key pointer.

Either INFILE or INDATASET may be used to specify the base cluster. INFILE identifies a DD statement that names the base cluster, while INDATASET names the base cluster to be located through the catalog.

Similarly, OUTFILE or OUTDATASET specify the target alternate index. Multiple alternate indexed may be loaded from the same base cluster with a single execution of the BLDINDEX command, so OUTFILE may specify multiple DD names and OUTDATASET may specify multiple alternate index entry names.

By default, an internal sort will be performed to place the alternate keys into ascending sequence. If there is insufficient virtual storage to perform the sort, an external sort will be performed automatically. If an external sort is required, two work files must be provided. The default DD names for these files is IDCUT1 and IDCUT2. These DD names may be overridden by the WORKFILES parameter.

Examples

The following jobstream uses the BLDINDEX command to build the alternate keys and load the alternate indexes defined by the previous example jobstream.

```
//BLDINDEX JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//TIMECLOK DD  DSN=PAYROLL.TIMECLOK.FILE,DISP=SHR
//TIMECLKX DD  DSN=PAYROLL.TIMECLOK.AIX,DISP=SHR
//IDCUT1   DD
VOL=SER=MVS802,UNIT=3380,DISP=OLD,AMP='AMORG'
//IDCUT2   DD
VOL=SER=MVS802,UNIT=3380,DISP=OLD,AMP='AMORG'
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* BUILD THE ALTERNATE INDEX FOR KSDS
MVS801.STUDENT.FILE */

  BLDINDEX INDATASET(MVS801.STUDENT.FILE)
-
           OUTDATASET(MVS801.STUDENT.AIX)

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(MVS801.STUDENT)

  /* BUILD THE ALTERNATE IX FOR ESDS
PAYROLL.TIMECLOK.FILE  */

  BLDINDEX INFILE(TIMECLOK)
-
           OUTFILE(TIMECLKX)

  IF LASTCC = 0 THEN
-
        LISTCAT ALL LEVEL(PAYROLL)

/*
//
```

The SYSOUT from this jobstream can be viewed as BLDINDEX. You can see that there are nonunique alternate key values for PAYROLL.TIMECLOK by comparing the REC-TOTAL entry under the STATISTICS section for the base cluster - PAYROLL.TIMECLOK.FILE.DATA (840 records) - and for the alternate index - PAYROLL.TIMECLOK.AIX.DATA (210 records).

At this point, there are two Key Sequenced clusters that contain data which is logically related. In order to be used, the clusters must be physically related, which requires the creation of a PATH.

Model syntax for the DEFINE PATH command:

DEFINE **PATH**
  (**NAME**(entryname)
  PATH**ENT**RY(entryname[/password])
  [**FILE**(ddname)]
  [**MODEL**(entryname[/password] [catname[/password]])]
  [**TO**(date) | **FOR**(days)]

  [**CAT**ALOG(catname[/password])]

Note that I have simplified the syntax diagram by excluding advanced optional parameters.

NAME specifies the name given to the path. PATHENTRY specifies the name of the alternate index cluster to which this path will point. When the path is created, accessing the VSAM object for the path will return the records from the base cluster in sequence by the values of the keys in the alternate index.

Examples

The following jobstream uses the DEFINE PATH command to build the paths for the alternate indexes built by the previous jobstream.

```
//DEFPATH  JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* BUILD THE PATH FOR MVS801.STUDENT.AIX
*/

  DEFINE PATH (
-
               NAME(MVS801.STUDENT.NAME.IX)
-
               PATHENTRY(MVS801.STUDENT.AIX) )

  IF LASTCC = 0 THEN
-
        LISTCAT ALL ENTRY(MVS801.STUDENT.NAME.IX)

  /* BUILD THE PATH FOR PAYROLL.TIMECLOK.AIX
*/

  DEFINE PATH (
-
               NAME(PAYROLL.TIMECLOK.EID.IX)
-
```

```
                 PATHENTRY(PAYROLL.TIMECLOK.AIX) )

  IF LASTCC = 0 THEN                                                  -
        LISTCAT ALL ENTRY(PAYROLL.TIMECLOK.EID.IX)

/*
//
```

The SYSOUT from this jobstream can be viewed as DEFPATH. By looking at the ASSOCIATIONS in the LISTCAT output, you can see that the PATH has related all component parts of the alternate index cluster and the base record cluster for each of the example sets.

To illustrate the effect of the alternate index, here is the SYSOUT from PRINT commands showing a segment of the data processed by way of both the base cluster and the path for each of the two example sets: AIXPRINT.

Defining Generation Data Groups

A generation data group is a group of related datasets, usually created in chronological order. Using a generation data group to control the creation and retention of this type of dataset group greatly simplifies data management. Before creating the first generation of a generation data group, two steps are required. First, the generation data group base, which contains the rules for managing the group, is defined using AMS. Second, a model must be created for the DCB parameters of the individual datasets to be created.

Model syntax for the DEFINE GENERATIONDATAGROUP command:

DEFINE **G**ENERATION**D**ATA**G**ROUP
   (**NAME**(entryname)
   LIMIT(limit)
   [**EMP**TY | **NOEMP**TY]
   [**SCR**ATCH | **NOSCR**ATCH]
   [**TO**(date) | **FOR**(days)])

   [**CAT**ALOG(catname[/password])]

LIMIT specifies the maximum number (from 1 to 255) of generations that are to be kept.

EMPTY specifies that **all** the generation datasets are to be uncataloged when the maximum is reached and another generation dataset is to be cataloged. NOEMPTY (the default) specifies that only the oldest generation dataset is to be uncataloged when the maximum is reached.

SCRATCH specifies that the generation dataset's entry is to be deleted from the Volume Table Of Contents when the generation dataset is uncataloged. The generation dataset ceases to exist. NOSCRATCH (the default) specifies that the generation dataset's entry is not to be removed from the VTOC when the
generation dataset is uncataloged.

Model DSCB

When generation datasets are created, the DCB information for the new dataset must come from a model dataset. The model dataset does not occupy any storage space, but exists only as a VTOC entry. The model must be on the same volume where the catalog in which the generation data group is cataloged resides. Since the name for the GDG base is already in the catalog, the model (which also has the same name) cannot be cataloged.

An alternative is to have a single model DSCB that is used for all generation datasets catalogued on a system pack, such as the System Residence. When a new dataset geneation is created, the model DSCB is specified plus overriding DCB parameters specific to the particular dataset being created.

Example

The following jobstream defines a Generation Data Group and also creates a model DSCB which can be used for the creation of the generation datasets.

```
//DEFGDG   JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//MODEL    DD  DSN=MVS501.ACH.TRANS,DISP=(NEW,KEEP),
//             UNIT=3350,VOL=SER=MVS501,SPACE=(TRK,0),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  /* DELETE GENERATION DATA GROUP, IF IT EXISTS
*/

  DELETE MVS501.ACH.TRANS GENERATIONDATAGROUP

  /* DEFINE GENERATION DATA GROUP
*/

  DEFINE GENERATIONDATAGROUP (
-
               NAME(MVS501.ACH.TRANS)
-
               LIMIT(5)
-
               SCRATCH )
  IF LASTCC = 0 THEN
-
        LISTCAT ALL ENTRIES(MVS501.ACH.TRANS)

/*
//
```

The SYSOUT from this jobstream can be viewed as DEFGDG. Note that the SPACE specified for the model is (TRK,0), so no space is reserved for the model. For an example of a generation data group where the LIMIT has been reached, I used the REPRO command to create a few successive generations of a dataset - GDGCOPY. In this job, generation #7 was cataloged. From the ASSOCIATIONS under the generation data group base entry in the catalog, you can see that there are five active generations being maintained.

Verify VSAM Dataset

If a job terminates abnormally and a VSAM dataset is not closed, the catalog entry for the dataset is flagged to indicate that the dataset may be corrupt. Before the dataset can be opened again, the VERIFY command must be used to correctly identify the end of the dataset and reset the catalog entry.

Model syntax for the VERIFY command:

VERIFY {**FILE**(ddname[/password]) |
  DATA**S**ET(entryname[/password])}

The dataset to be verified may be specified by either FILE, which designates a DD name for which a DD statement must be provided to identify the dataset, or DATASET, which designates the name which is used to identify the dataset in the catalog. There is no CATALOG option for the VERIFY command, so if there is no alias defined for the catalog in which the dataset can be located, it will be necessary to use either a JOBCAT or STEPCAT DD statement for the proper catalog.

Example

The following jobstream verifies a VSAM dataset.

```
//VERIFY   JOB 'JAY
MOSELEY',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=A
//IDCAMS  EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *

  VERIFY DATASET(MVS801.STUDENT.FILE)

/*
//
```

The SYSOUT from this jobstream can be viewed as VERIFY.