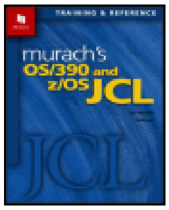

Murach's OS/390 and z/OS JCL

by Raul Menendez and Doug Lowe

Table of Contents

Chapter 16: How to Use Access Method Services (AMS)

In this chapter, you'll learn how to use the VSAM utility program called Access Method Services, or AMS. As you will see, it lets you define VSAM data sets with all of the possible options. It lets you define and build alternate indexes. It lets you print and modify catalog entries. And it lets you perform other useful functions like deleting, printing, and copying data sets. In short, AMS is a valuable program that every programmer should know how to use.

How to code AMS jobs

To start, this chapter introduces you to the AMS commands you're most likely to come across in your shop. Then, it shows you how to code the JCL for a job that uses AMS, and it shows you how to code the AMS commands within the job stream.

Some commonly−used AMS commands

Figure 16−1 summarizes the AMS commands that are most commonly used in an OS/390 shop. In this chapter, you'll learn how to use all of the commands listed for the application programmer. These are the commands that every application programmer is likely to need at one time or another, so you should be familiar with all of them. As you can see, these include commands for defining all three types of VSAM data sets, for defining and building alternate indexes for key−sequenced data sets, and for listing catalog information.

The other commands that are listed in this figure are for functions that are usually handled by the systems programmer. These include commands for defining the master catalog and user catalogs, for defining aliases for user catalogs, and for backing up and restoring data sets. Although this chapter doesn't present their coding details, you should be able to master these and other AMS commands on your own once you complete this chapter, just by consulting the appropriate reference manual.

Please note that the commands listed in this figure are all *functional commands*. In addition, AMS provides *modal commands* that you can use for conditional processing of functional commands. Here again, the modal commands aren't presented in this chapter, but you should be able to master them on your own without much difficulty.

As you read the rest of this chapter, you'll see that more detail is presented for each command than you can possibly remember. However, all of the information in the figures is presented in an efficient reference format. As a result, you can do a first reading of this chapter with the goal of understanding what each command can do. Then, when you need to use one of the commands, you can refer back to the appropriate figures.

Figure 16−1: Some commonly−used AMS commands

AMS commands for the application programmer

AMS command	Function
DEFINE CLUSTER	Defines a VSAM data set, whether it's key−sequenced, entry−sequenced, or relative−record.
LISTCAT	Lists information about data sets.
ALTER	Changes the information that has been specified for a catalog, cluster, alternate index, or path.
DELETE	Removes a catalog entry for a catalog, cluster, alternate index, or path.
PRINT	Prints the contents of a VSAM or non−VSAM data set.

REPRO	Copies records from one VSAM or non–VSAM data set to another VSAM or non–VSAM data set.
DEFINE ALTERNATEINDEX	Defines an alternate index.
DEFINE PATH	Defines the path that relates an alternate index to its base cluster.
BLDINDEX	Builds an alternate index.

AMS commands for the systems programmer

AMS command	Function
DEFINE MASTERCATALOG	Defines a master catalog.
DEFINE USERCATALOG	Defines a user catalog.
DEFINE ALIAS	Assigns an alternate name to a user catalog.
EXPORT	Produces a backup or transportable copy of a data set.
IMPORT	Restores an exported copy of a data set.

Description

- The commands listed for the application programmer are the ones that are presented in this chapter. These are the *functional commands* that most application programmers will need to use.
- The commands listed for the systems programmer are the main functional commands that they need to use. In addition, there are *modal commands* like IF and SET commands that provide for conditional processing of functional commands.

The JCL requirements for using AMS commands

Figure 16–2 presents the JCL requirements for using AMS commands. To start, you need to code an EXEC statement for the program named *IDCAMS* (you'll hear it pronounced I–D–cams or *id*–cams). Because IDC is the standard prefix for VSAM programs, this is simply the name of the AMS program. Then, you must code a SYSPRINT DD statement for the printed output produced by the program, and a SYSIN DD statement for the AMS commands that are submitted to the program.

Usually, you can code the SYSPRINT and SYSIN DD statements as shown in the two examples. Here, the SYSPRINT statement says that the output class is the same as the message class in the JOB statement. And the SYSIN statement says that the input will follow as an instream data set. Then, the data that follows (one or more AMS commands) is ended by either a /* statement or the next JCL statement.

In some cases, you may also have to code DD statements for the data sets that are going to be processed by an AMS command, but that's relatively uncommon. You'll see how that works later in this chapter.

The syntax requirements for AMS commands

Figure 16–2 also presents the syntax requirements for the AMS commands that you include in the SYSIN data set. If you study these requirements and the examples, you should be able to quickly master the syntax. Just be careful to start every command line in column 2 or later (not column 1) and to end every continued line with a hyphen (–).

If you want to use comments within your commands, you can code them as shown in the second example in this figure. Note that the hyphen for a continued line that ends with a comment must go after the comment.

Figure 16–2: The JCL and AMS syntax requirements for an AMS job

The JCL for an AMS job that runs a LISTCAT command

```
//MM01LC1  JOB  (36512),'R MENENDEZ',NOTIFY=&SYSUID
//         EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 LISTCAT ENTRIES(MM01.CUSTOMER.MASTER) -
         VOLUME
/*
```

The JCL for an AMS job with comments that runs an ALTER command

```
//MM01LC2  JOB  (36512),'R MENENDEZ',NOTIFY=&SYSUID
//         EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 /* THIS JOB CHANGES BOTH THE NAME AND FREESPACE ALLOCATION */
 ALTER MM01.CUSTOMER.MASTER -
       NEWNAME(MM01.CUSTMAST)    /* CHANGE NAME */ -
       FREESPACE(10 10)          /* CHANGE FREESPACE ALLOCATION */
/*
```

The JCL requirements for an AMS job

- The program name for Access Method Services (AMS) is *IDCAMS*.
- A SYSPRINT DD statement is required for the printed output produced by IDCAMS.
- A SYSIN DD statement is required for the commands that are submitted to IDCAMS. Usually, this data set is an instream data set.
- If you run a command that causes a data set to be processed, you may also have to include a DD statement for that data set. Often, though, the data set is identified by the command so this isn't necessary.

The AMS syntax requirements

- AMS commands must be coded within columns 2 through 72, so don't start a command in column 1.
- To continue a command on the next line, end the current line with one or more spaces followed by a hyphen, then continue coding on the next line.
- To separate parameters and subparameters, you can use spaces or commas. In this book, spaces are used as the separators.

- To code a comment, start it with /* and end it with */. Comments can be coded on separate lines or to the right of a command line. When they are coded on the right, the hyphen for a continuation comes after the comment.

- Although you can code more than one parameter on each line, we recommend that you code one parameter per line so it's easier to read and modify the command.

How to use the DEFINE CLUSTER command to define a data set

To create a VSAM data set when you're using AMS, you use the DEFINE CLUSTER command. To make this command more manageable, the syntax summary in figure 16–3 presents just the parameters that the application programmer is likely to need. Other parameters that can affect performance are omitted because they are normally used by the systems programmer when a data set is put into production.

If you study the syntax for this command, you can see that some of its parameters can be coded at three different levels. The cluster level applies to the entire data set. The data level applies to the data component of a data set. And the index level applies to the index component of a KSDS. As you read about these parameters in the pages that follow, you'll learn what parameters should be coded at what levels.

The good news is that this is the most complicated of the AMS commands, so the rest of this chapter should be relatively easy once you understand how this command works. To make the many parameters easier to understand, they are grouped by function in the next five figures. As you read about the individual parameters, you can refer back to this syntax summary to see where they fit.

Figure 16–3: The syntax of the DEFINE CLUSTER command

The syntax of the DEFINE CLUSTER command

```
DEFINE CLUSTER (   NAME(entry-name)
                 [ OWNER(owner-id) ]
                 [ FOR(days) | TO(date) ]
                 [ INDEXED | NONINDEXED | NUMBERED | LINEAR]
                 [ RECORDSIZE(avg max) ]
                 [ KEYS(length offset) ]
                 [ SPANNED | NONSPANNED ]
                 [ CISZ(size) ]
                 [ FREESPACE(ci ca) ]
                 [ VOLUMES(vol-ser…) ]
                 [ FILE(ddname) ]
                 [ {CYLINDERS(primary [secondary])}
                   {KILOBYTES(primary [secondary])}
                   {MEGABYTES(primary [secondary])}
                   {TRACKS(primary [secondary])    }
                   {RECORDS(primary [secondary])   } ]
                 [ REUSE | NOREUSE ]
                 [ SHAREOPTIONS(a b) ]
                 [ IMBED | NOIMBED ]      )
                 [ STORAGECLASS(storage-class) ]
                 [ DATACLASS(data-class) ]
                 [ MANAGEMENTCLASS(management-class) ] )
 [ DATA        ( [ NAME(entry-name) ]
                 [ VOLUMES(vol-ser…) ]
                 [ FILE(ddname) ]
                 [ CISZ(size) ]
                 [ {CYLINDERS(primary [secondary])}
                   {KILOBYTES(primary [secondary])}
                   {MEGABYTES(primary [secondary])}
                   {TRACKS(primary [secondary])   }
                   {RECORDS(primary [secondary])  } ] ) ]
 [ INDEX       ( [ NAME(entry-name) ]
                 [ VOLUMES(vol-ser…) ]
                 [ {CYLINDERS(primary [secondary])}
                   {KILOBYTES(primary [secondary])}
                   {MEGABYTES(primary [secondary])}
                   {TRACKS(primary [secondary])   }
                   {RECORDS(primary [secondary])  } ] ) ]
 [ CATALOG(name) ]
```

Description

- The DEFINE CLUSTER command is used to define a KSDS, ESDS, or RRDS. The parameters above are the ones the application programmer is most likely to use.
- Parameters at the CLUSTER level apply to the entire cluster. Parameters at the DATA or INDEX level apply only to the data or index component.

Parameters that identify a data set

Figure 16–4 summarizes three parameters that identify a data set. Of these, the most important one is the NAME parameter. If you refer back to figure 16–3, you can see that it can be coded at the cluster, data, and index level. Because VSAM creates cryptic names for the data and index components if you only code the NAME parameter at the cluster level, I recommend that you also code this parameter at the lower levels.

In the examples, you can see a simple naming convention for the three levels. At the cluster level, you name the data set using the standard OS/390 naming rules. Then, at the data and index levels, you add DATA and INDEX to the cluster name. That makes it easy for you to identify the components of a data set when you list catalog entries later on.

In contrast, the OWNER parameter is used for documentation only, so it doesn't affect the function of this or subsequent AMS commands. In the second example, you can see that this parameter is coded so MM01 will be used as the owner–id instead of the TSO user–id.

If you refer back to figure 16–3, you can see that the CATALOG parameter is the only one that isn't coded at the cluster, data, or index level. So if you use it, you always code it as the last parameter of the command. Normally, though, you don't need to code it because the high–level qualifier of the cluster name is used to identify the catalog that owns it.

Note, however, that if the high–level qualifier isn't defined as a user catalog or an alias, OS/390 uses the step catalog, the job catalog, or the master catalog as the catalog for the data set. This is the standard search sequence that is used for identifying the catalog to be used. If that isn't what you want, you do need to code the CATALOG parameter. In the second example in this figure, though, the CATALOG parameter isn't necessary because MM01 is also the high–level qualifier of the cluster name.

Figure 16–4: Parameters that identify a data set

The parameters for data set identification

Parameter	Explanation

NAME(entry−name)	Specifies the name of the cluster or component.
OWNER(owner−id)	Specifies a one− to eight−character owner−id. This is for documentation only. If you omit it when you issue a DEFINE CLUSTER command from a TSO terminal, VSAM uses your TSO user−id as the owner−id.
CATALOG(name)	Specifies the name of the catalog that will own the cluster. If omitted, OS/390 uses its standard search sequence to identify the catalog.

OS/390's standard search sequence for a catalog entry

Catalog with matching high−level qualifier → Step catalog → Job catalog → Master catalog

A command that defines a KSDS

```
DEFINE CLUSTER  ( NAME(MM01.CUSTOMER.MASTER)          -
                  INDEXED                             -
                  RECORDSIZE(200 200)                 -
                  KEYS(6 0)                           -
                  VOLUMES(TSO001) )                   -
       DATA     ( NAME(MM01.CUSTOMER.MASTER.DATA)     -
                  CYLINDERS(50 5)                     -
                  CISZ(4096) )                        -
       INDEX    ( NAME(MM01.CUSTOMER.MASTER.INDEX) )
```

A command that defines an ESDS

```
DEFINE CLUSTER  ( NAME(MM01.AR.TRAN)                  -
                  OWNER(MM01)                         -
                  NONINDEXED                          -
                  RECORDSIZE(190 280)                 -
                  CISZ(4096)                          -
                  VOLUMES(TSO001) )                   -
       DATA     ( NAME(MM01.AR.TRAN.DATA)             -
                  CYLINDERS(10 1) )                   -
       CATALOG(MM01)
```

Description

- When you code a value for the NAME parameter at the cluster level but not at the DATA or INDEX level, VSAM creates cryptic names for the data and index components. That's why we recommend that you code your own names at all three levels.

- The CATALOG parameter is the only one that isn't coded at the CLUSTER, DATA, or INDEX level. If you code it, it should be the last parameter for the command, but it usually isn't necessary because the standard search sequence identifies the appropriate catalog.

Parameters that describe the data set's characteristics

Figure 16−5 summarizes the parameters that describe a data set's characteristics. For instance, the first parameter indicates what type of file organization is used for the data set: KSDS, ESDS, or RRDS. The KEYS parameter gives the length and starting position of the primary key for a KSDS. And the SPANNED parameter indicates whether a record can flow from one control interval into the next one, which usually isn't allowed.

The other three parameters set the record size, the control interval size, and the amount of free space for each control interval and control area. If you omit them, VSAM will set default values for them, but that's usually not what you want. Since these parameters can have a significant effect on performance, they should be given special attention.

In the RECORDSIZE parameter, you code the average and maximum record sizes. For fixed−length records, you code the same value for both as shown in the first example. For variable−length records, you code an estimate for the average record length and the maximum record size as shown in the second example. The only time the average record size really matters is when you specify the space allocation in records.

In the CISZ parameter, you code the size of the control intervals for the data set. As this figure says, 4096 will work for most data sets. But if the maximum record size is larger than that, you can adjust the control interval size to the next multiple of 2K that is larger than the maximum record size to avoid the need for spanned records. These simple rules will work for test data sets, and the systems programmer can adjust this size for improved performance when the data set is put into production.

For an ESDS or RRDS, you can code the CISZ parameter at the cluster or the data level. For a KSDS, though, you should code this parameter at the data level, and let VSAM set an efficient control interval size for the index component. If you code this parameter at the cluster level for a KSDS, VSAM uses that control interval size for both the data and the index components.

In the FREESPACE parameter, you code the percentage of free space that you want left in each control interval and each control area when the file is created. So the free space allocation in the first example provides that 30% of each control interval and 20% of each control area be left empty. Since the default is no free space, this is an important parameter for the performance of any data set that receives file additions. When a data set is put into production, this parameter should be set based on the frequency of additions and how often the data set is reorganized.

Note that the free space is taken from, not added to, the amount of space that is allocated to the file. So if you provide for 20% free space, you may need to increase the space allocation for the data set accordingly.

Figure 16−5: Parameters that describe the data set's characteristics

The parameters for data set characteristics

Parameter	Explanation
{INDEXED }	Specifies that a KSDS is being defined.
{NONINDEXED}	Specifies that a ESDS is being defined.
{NUMBERED }	Specifies that an RRDS is being defined.
RECORDSIZE(avg max)	Specifies the average and maximum record size. If omitted, VSAM sets defaults of 4089 for both average and maximum if the records aren't SPANNED; it sets defaults of 4086 and 32600 if the records are SPANNED.
KEYS(len off)	Specifies the length and offset of the primary key for a KSDS. The default length is 64 bytes with an offset of 0.
SPANNED|NONSPANNED	Specifies whether records can cross control interval boundaries. If the record size is larger than one control interval, you need to code SPANNED.
CISZ(size)	Specifies the size of the control intervals. For most data sets, you can set this at 4096 unless the records are larger than that. For larger records, you can set this at the next multiple of 2K (2048), like 6144 or 8192. If omitted, VSAM sets this size.
FREESPACE(ci ca)	Specifies the percentage of free space to reserve for the control intervals (ci) and control areas (ca) of the data component. If file additions are likely,

	you should code this parameter because the default is no free space. The free space that you specify is part of the space allocated to the data component.

A command that defines a KSDS with fixed−length records and free space

```
DEFINE CLUSTER  ( NAME(MM01.CUSTOMER.MASTER)          -
                  INDEXED                             -
                  RECORDSIZE(600 600)                 -
                  KEYS(6 0)                           -
                  VOLUMES(TSO001)                     -
                  FREESPACE(30 20) )                  -
       DATA     ( NAME(MM01.CUSTOMER.MASTER.DATA)     -
                  CYLINDERS(50 5)                     -
                  CISZ(4096) )                        -
       INDEX    ( NAME(MM01.CUSTOMER.MASTER.INDEX) )
```

A command that defines an ESDS with variable−length records

```
DEFINE CLUSTER  ( NAME(MM01.AR.TRAN)                  -
                  NONINDEXED                          -
                  RECORDSIZE(400 800)                 -
                  CISZ(4096)                          -
                  VOLUMES(TSO001) )                   -
       DATA     ( NAME(MM01.AR.TRAN.DATA)             -
                  CYLINDERS(10 1) )
```

Description

- The control interval size and amount of free space specified for a KSDS can have a significant effect on performance. For production data sets, then, the CISZ and FREESPACE parameters are usually set by a systems programmer.

Parameters that specify the data set’s space allocation

Figure 16−6 summarizes the parameters for the space allocation of a data set. Two that you’re already familiar with are the VOLUMES and space allocation parameters, so let’s start with them and end with IMBED and FILE.

The VOLUMES parameter is required, which means that AMS doesn’t provide for a non−specific volume request (although JCL does). When you code this parameter, you can specify more than one volume separated by spaces. In most cases, you code this parameter at the cluster level so the data and index components of a KSDS are on the same volume.

The space allocation parameter for the DEFINE CLUSTER command is coded in much the same way that you code the SPACE parameter in the DD statement for a non−VSAM file. First, you code the allocation unit. Then, you code the primary space allocation for the data set followed by the secondary allocation that is used when the primary allocation is filled up. Up to 122 of these secondary allocations can be used before the data set has to be reorganized.

When you allocate the space in terms of cylinders, the control area for the data set will be one cylinder. Because this leads to an efficient index structure for key−sequenced data sets, you’ll use this approach most often. However, if you don’t know the size of a cylinder on the device being used, the easiest way to specify the space allocation is in terms of records. Then, VSAM allocates the minimum number of tracks or cylinders that are required for the allocation. You can also allocate the space in terms of megabytes, kilobytes, or tracks. In any case, the systems programmer can adjust the value for production performance.

If you refer back to figure 16–3, you can see that the space allocation can be coded at the cluster, data, or index level. When you define an ESDS or RRDS, it doesn't matter whether you code the space allocation at the cluster or data level. Either way, the entire allocation is given to the data component.

When you define a KSDS, though, there is a slight difference in how the allocations are made. If you code the allocation at the data level, the entire allocation is given to the data component, and an additional allocation that's large enough for the index is given to the index component. But if you code the allocation at the cluster level, VSAM determines how much space to give to the index component and gives the rest to the data component. Either way, VSAM does a good job of calculating how much space is required for the index component, so you don't need to override that calculation by coding a space allocation at the index level.

If you code the IMBED parameter, the sequence set records of the index component are imbedded in the first tracks of the control areas, which can improve performance in some cases. But whether that's coded should probably be left to the systems programmer.

If you code the FILE parameter, it means that you're using an old DASD that has removable disk packs. Then, when you run a job that processes the data set, you need to supply a DD statement with the ddname that's given in the FILE parameter. That DD statement should identify the volumes that are to be used.

Figure 16–6: Parameters that specify the data set's space allocation

The parameters for space allocation

Parameter	Explanation
VOLUMES(vol–ser)	Specifies one or more volumes that will contain the cluster or component. It is required, and it's normally coded at the cluster level of a KSDS so the index and data components are on the same volume.
FILE(ddname)	Specifies a ddname that identifies a DD statement that allocates the volume or volumes when the job is run. This is required only for removable DASD volumes.
unit (primary [secondary])	*Primary* specifies the amount of space to allocate initially, expressed in terms of the *unit* (cylinders, kilobytes, megabytes, tracks, or records). *Secondary* specifies the secondary space allocation. If you specify this allocation in cylinders, the control areas will be one cylinder each.
IMBED|<u>NOIMBED</u>	Specifies whether sequence set records should be imbedded in the data component of a KSDS. When imbedded, the first track of each control area is used to store the sequence set records, which can improve performance.

A command that defines a KSDS with the space allocation at the data level

```
DEFINE CLUSTER  ( NAME(MM01.CUSTOMER.MASTER)          -
                  INDEXED                             -
                  RECORDSIZE(600 600)                 -
                  KEYS(6 0)                           -
                  FREESPACE(30 20)                    -
                  VOLUMES(TSO001)                     -
                  IMBED )                             -
       DATA     ( NAME(MM01.CUSTOMER.MASTER.DATA)     -
                  CYLINDERS(50 5)                     -
                  CISZ(4096) )                        -
       INDEX    ( NAME(MM01.CUSTOMER.MASTER.INDEX) )
```

What unit to use for space allocation

- For most data sets, you should allocate space in terms of cylinders so the control areas will be one cylinder each. For a KSDS, that provides for an efficient index structure.

- If you don't know the cylinder size for the device that will be used for a data set, it's usually easiest to specify the space allocation in terms of records. As an alternative, you can figure the total space required and then specify the allocation in terms of kilobytes or megabytes.

Where to code the space allocation for a KSDS

- If you specify the allocation at the cluster level, VSAM determines how much space to allocate for the index component and allocates the rest to the data component.
- If you specify the allocation at the data level, VSAM allocates the full amount to the data component and allocates an additional amount to the index component.
- If you specify an allocation at the index level, you override VSAM's calculation for how much index space is required. Normally, though, you don't need to do that.

Parameters for SMS−managed files

If you've read chapter 13, you know how SMS works and you know about the three DD parameters that you can use to assign classes to SMS−managed files. You can also assign classes to a file when you define a cluster with AMS by using the three parameters in figure 16−7. If you omit one of these parameters, the SMS automatic class selection (ACS) routines assign the appropriate class to the file.

In the first example in this figure, none of the parameters are coded, so the ACS routines select the classes for the file, and the classes set all of the other parameters for the data set. As a result, only the NAME parameter is coded at the cluster, data, and index levels.

In the second example, the parameters for the storage and data classes are coded, so just the management class is assigned by the ACS routines. In addition, other parameters are coded that override some of the parameters that are set by the classes.

Before you can use the SMS classes with confidence, of course, you must find out what parameter values each class assigns. Once you've done that, though, you can use the classes to reduce the number of parameters that you have to code when you define a data set. Note, too, that you can use the DATACLASS parameter to define any data set, not just an SMS−managed data set, as long as SMS is active on your system.

Figure 16−7: Parameters for SMS−managed files

The parameters for SMS−managed files

Parameter	Explanation
STORAGECLASS	Specifies the storage class for SMS−managed data sets only.
DATACLASS	Specifies the data class for any data set as long as SMS is installed and active.
MANAGEMENTCLASS	Specifies the management class for SMS−managed data sets only.

A command that defines a KSDS with the SMS defaults

```
DEFINE CLUSTER  ( NAME(MM01.CUSTOMER.MASTER) )          -
       DATA     ( NAME(MM01.CUSTOMER.MASTER.DATA) )     -
       INDEX    ( NAME(MM01.CUSTOMER.MASTER.INDEX) )
```

A job that overrides two classes, the key location, and the space allocation

```
DEFINE CLUSTER  ( NAME(MM01.CUSTOMER.MASTER)            -
                  STORAGECLASS(MVPS100)                 -
                  DATACLASS(MVPD050)                    -
                  KEYS(6 0) )                           -
       DATA     ( NAME(MM01.CUSTOMER.MASTER.DATA)       -
                  CYLINDERS(50 5) )                     -
       INDEX    ( NAME(MM01.CUSTOMER.MASTER.INDEX) )
```

Description

- Chapter 13 shows how to use the SMS facilities with JCL, and you can also use these facilities with the DEFINE CLUSTER command.
- SMS manages files by establishing storage classes, data classes, and management classes for different types of data sets. These classes determine what volume a data set is written to, what the characteristics of the data set are, how long it will reside on the volume before it is archived offline, and how often it should be backed up.
- If you don't use the AMS parameters to specify the storage, data, and management class for a cluster, the Automatic Class Selection (ACS) routines that are set up for a system automatically pick these classes.
- If necessary, you can override the parameters in a class by coding the parameters in the AMS command.

Other DEFINE CLUSTER parameters

Figure 16–8 presents the last set of parameters that an application programmer is likely to code in the DEFINE CLUSTER command. If you code the FOR or TO parameter, you set the *retention period* or *expiration date* that determines how long the data set should be retained before it is deleted. For instance, the first example in this figure specifies a retention period of 30 days.

In general, it's better to use the FOR parameter than the TO parameter because a retention period is easier to code than an expiration date. When you use a retention period, VSAM converts it to an expiration date anyway, so in either case, it's an expiration date that gets stored in the catalog entry for the data set.

As you might guess, the expiration date determines when a data set can be deleted. However, you can override the date using a parameter on the DELETE command, as you'll see later in this chapter. In contrast, if you omit both the FOR and the TO parameters, a data set can be deleted at any time.

If you code the REUSE parameter as shown in the second example in this figure, you create a reusable data set. Then, if the data set is opened for input or for input and output, the data set is processed as usual. But when the data set is opened as output, the old records are ignored, so the new records overwrite them. The alternative to using a reusable data set is to delete the old data set, redefine it, and then recreate it.

The last parameter in this figure is the SHAREOPTIONS parameter. It tells VSAM what level of file sharing should be allowed for the data set. When you code this parameter, the first subparameter provides a value for the *cross−region share option*, which controls how two or more jobs on a single system can share a data set. The second subparameter provides a value for the *cross−system share option*, which controls how jobs on different processors can share a data set.

This figure also summarizes the values that you can use for these options. For most data sets, you'll use 1, 2, or 3 as the cross−region share option and 3 as the cross−system share option because 4 is too restrictive for either option.

Figure 16−8: Other DEFINE CLUSTER parameters

Other parameters of the DEFINE CLUSTER command

Parameter	Explanation
FOR(days) | TO(date)	Specifies a retention period (in the format dddd) or an expiration date (in the format yyyyddd).
REUSE|NOREUSE	Specifies whether a data set is reusable.
SHAREOPTIONS(a b)	Specifies the level of file sharing permitted as summarized below. The default level is 1 3.

Share options

Cross−region share options (a)

1. The file can be processed simultaneously by multiple jobs as long as all jobs open the file for input only. If a job opens the file for output, no other job can open the file.
2. The file can be processed simultaneously by multiple jobs as long as only one job opens the file for output; all other jobs must open the file for input only.
3. Any number of jobs can process the file simultaneously for input or output, but VSAM does nothing to insure the integrity of the file.
4. Any number of jobs can process the file simultaneously for input or output, but VSAM imposes these restrictions: (1) direct retrieval always reads data from disk even if the desired index or data records are already in a VSAM buffer; (2) data may not be added to the end of the file; and (3) a control area split is not allowed.

Cross−system share options (b)

3. Any number of jobs on any system can process the file simultaneously for input or output, but VSAM does nothing to insure the integrity of the file.
4. Any number of jobs on any system can process the file simultaneously for input or output, but VSAM imposes the same restrictions as for cross−region share option 4.

A command that defines a KSDS with share options and a retention period

```
DEFINE CLUSTER  ( NAME(MM01.CUSTOMER.MASTER)          -
                  INDEXED                             -
                  RECORDSIZE(800 800)           -
                  KEYS(6 0)                     -
                  VOLUMES(TSO001)               -
                  SHAREOPTIONS(2 3)             -
                  FOR(30) )                     -
       DATA     ( NAME(MM01.CUSTOMER.MASTER.DATA)      -
                  CYLINDERS(50 5)               -
                  CISZ(4096) )                  -
       INDEX    ( NAME(MM01.CUSTOMER.MASTER.INDEX) )
```

A command that defines a reusable ESDS

```
DEFINE CLUSTER  ( NAME(MM01.AR.TRAN)            -
                  NONINDEXED                    -
                  RECORDSIZE(400 800)           -
                  VOLUMES(TSO001)               -
                  REUSE )                       -
       DATA     ( NAME(MM01.AR.TRAN.DATA)       -
                  CYLINDERS(10 1) )
```

How to use the LISTCAT command to print catalog information

The LISTCAT command is the AMS command that lets you get data set information from a user catalog, which is something that you will frequently want to do. The syntax of this command is given in figure 16–9. As you can see, the parameters for this command let you identify the catalog you want to work with, the names of the entries to be listed, the types of entries to be listed, and the amount of information to be listed for each entry.

When you use this command, you want to be sure that you don't get far more information than you need. To do that, you try to restrict the number of entries that are listed, and you try to restrict the amount of information that is listed for each entry. In the example in this figure, the LISTCAT command lists just the NAME information for three named entries. In the next two figures, you'll learn more about specifying the entries and information that is listed.

If necessary, you can code the CATALOG parameter to identify the catalog that you want to work with. If you omit that parameter, though, VSAM uses the standard search sequence to determine the catalog that should be used: the catalog identified by the high–level qualifier of the entries listed, the step catalog, the job catalog, or the master catalog.

Figure 16–9: The syntax of the LISTCAT command

The syntax of the LISTCAT command

```
LISTCAT [ CATALOG(name) ]
        [ {ENTRIES(entry-name…)}
          {LEVEL(level)} ]
        [ entry-type… ]
        [ NAME | HISTORY | VOLUME | ALLOCATION | ALL ]
```

Explanation

CATALOG	Specifies the name of the catalog from which entries are to be listed. If omitted, the standard search sequence is used.
ENTRIES	Specifies the names of the entries you want to list.
LEVEL	Specifies one or more levels of qualification. Any data sets whose names match those levels are listed.
entry–type	Specifies the type of entries you want listed (you can code more than one type): ALIAS ALTERNATEINDEX or AIX

	CLUSTER DATA GENERATIONDATAGROUP or GDG INDEX NONVSAM PAGESPACE PATH USERCATALOG
NAME	Specifies that only the names and types of the specified entries be listed. This is the default.
HISTORY	Specifies that the NAME information plus the history information (such as creation and expiration dates) be listed.
VOLUME	Specifies that the HISTORY information plus the volume locations of the specified entries be listed.
ALLOCATION	Specifies that the VOLUME information plus detailed extent information be listed.
ALL	Specifies that all catalog information be listed.

A LISTCAT command that lists the NAME information for three data sets

```
LISTCAT CATALOG(MM01) -
        ENTRIES(MM01.CUSTOMER.MASTER -
                MM01.EMPLOYEE.MASTER -
                MM01.DAILY.TRANS)    -
        NAME
```

Description

- The LISTCAT output is written to the SYSPRINT file that's included in an AMS job.
- If an ENTRIES or LEVEL parameter isn't coded, VSAM lists *all* of the entries in the catalog. You should avoid that, though, because it may contain hundreds of entries.
- In general, you should try to make your LISTCAT commands as specific as possible by limiting the entry names and types as well as the type of information that's returned.

How to identify the entries you want to list

Figure 16–10 shows four ways to identify the entries that you want the LISTCAT command to list. In the first example, a *generic entry name* is used in the ENTRIES parameter to identify all entries that have MM01 as the high–level qualifier, any characters (*) at the next level, and MASTER at the third level. This will select entries with names like these:

```
MM01.CUSTOMER.MASTER
MM01.EMPLOYEE.MASTER
MM01.VENDOR.MASTER
```

Since you can code the asterisk at more than one level, this is an excellent way to identify the data sets that you want listed.

In the second example, you can see that using the LEVEL parameter is similar to using a generic entry name. In this parameter, you code a partial name that consists of one or more levels. VSAM then lists all of the catalog entries whose names begin with the partial name. This example, then, will list entries like:

```
MM01.CUSTOMER.MASTER
```

```
MM01.CUSTOMER.MASTER.DATA
MM01.CUSTOMER.MASTER.INDEX
MM01.CUSTOMER.TRANS
```

In other words, VSAM prints all entries with names that start with the levels specified in the LEVEL parameter, no matter how many levels the names contain.

In the third example, you can see how the entry−type parameter is used to restrict the entries that are listed. Here, just entries that are part of a generation data group (see chapter 12) are listed. But note that the LEVEL command also specifies that the high−level qualifier must be MM01.

Finally, in the fourth example, you can see that two (or more) entry−type parameters can be coded in a single command. In this case, entries for both data and index components will be listed.

Keep in mind that you should always limit the number of entries for a LISTCAT command by coding one of the three parameters in this figure. If you don't, the command will list all of the entries in the catalog, and there could be hundreds of them.

Figure 16−10: How to identify the entries you want to list

A LISTCAT command that uses a generic entry name

```
LISTCAT CATALOG(MM01)          -
        ENTRIES(MM01.*.MASTER)  -
        NAME
```

A LISTCAT command that uses the LEVEL parameter

```
LISTCAT LEVEL(MM01.CUSTOMER)    -
        NAME
```

A LISTCAT command that uses the LEVEL and entry−type parameters

```
LISTCAT LEVEL(MM01)              -
        GDG                      -
        NAME
```

A LISTCAT command that uses two entry−type parameters

```
LISTCAT CATALOG(MM01)            -
        DATA                     -
        INDEX                    -
        NAME
```

Description

- To code a *generic entry name*, you replace one or more of the levels in a data set name with an asterisk. The asterisk means that any characters will be accepted for that level.
- When you code the LEVEL parameter, you specify a partial data set name consisting of one or more levels. Then, all data set names that begin with those levels will be returned.
- If you don't specify an ENTRIES, LEVEL, or entry−type parameter, AMS lists all of the entries in the catalog.

How to limit the information that's listed

Figure 16–11 shows the parameters that determine what information the LISTCAT command lists for each entry. The NAME parameter lists the least information for each entry, while the ALL parameter lists all of the information for each entry.

To give you some idea of what information is included with each parameter, this figure shows some output for the NAME parameter and some for the VOLUME parameter. For the NAME parameter, you can see that the information includes just the type, entry name, and owning catalog. In this case, ICFCAT.VSTOR02 is the actual name of the catalog, while MM01 is the alias.

For the VOLUME parameter, you can see the HISTORY information plus the VOLUME information that is added to the NAME information. Here, the history information includes the owner–id, the VSAM release that the data set was created under, and the creation and expiration dates for the entry. As for the volume information, it includes the volume serial numbers of the volumes used for the data set as well as a hex code for the device type.

If you specify ALLOCATION instead of VOLUME, you get the volume information plus information about the disk extents used for each entry. And if you specify ALL, you get the allocation information plus attribute information and statistics. Although examples aren't included in this book, you can easily make your own by running the LISTCAT command for one of the data sets on your system.

If you've read about generation data groups in chapter 12, you may now be interested to look at the last entry for the NAME output in this figure. Here, you can see the full name of one of the generations of a non–VSAM data set:

```
MM01.DAILYEMS.G0930V00
```

In this case, the name of the data set is MM01.DAILYEMS, and the generation number is G0930V00. So if you want to process one of the generations of a data set and need to know its generation number, using the LISTCAT command is one way to get it.

Figure 16–11: How to limit the information that's listed

The five information types

```
NAME    HISTORY    VOLUME    ALLOCATION    ALL
```

Example of NAME output

```
 LISTCAT LEVEL(MM01) -
         NAME

CLUSTER ——— MM01.CUSTOMER.MASTER
     IN-CAT — ICFCAT.VSTOR02

DATA ————— MM01.CUSTOMER.MASTER.DATA
     IN-CAT — ICFCAT.VSTOR02

INDEX ———— MM01.CUSTOMER.MASTER.INDEX
     IN-CAT — ICFCAT.VSTOR02

GDG BASE ——— MM01.DAILYEMS
     IN-CAT — ICFCAT.VSTOR02

NONVSAM ——— MM01.DAILYEMS.G0930V00
     IN-CAT — ICFCAT.VSTOR02
```

Example of VOLUME output

```
 LISTCAT ENTRIES(MM01.CUSTOMER.MASTER) -
         VOLUME

CLUSTER ——— MM01.CUSTOMER.MASTER
     IN-CAT — ICFCAT.VSTOR02
     HISTORY
```

```
      OWNER-IDENT-----MM01      CREATION-----2002.129
      RELEASE---------2     EXPIRATION----2002.365

  DATA --- MM01.CUSTOMER.MASTER.DATA
    IN-CAT - ICFCAT.VSTOR02
    HISTORY
      OWNER-IDENT---(NULL)      CREATION-----2002.129
      RELEASE---------2     EXPIRATION----2002.365
    VOLUMES
      VOLSER-------TSO001     DEVTYPE---X'3010200F'

  INDEX --- MM01.CUSTOMER.MASTER.INDEX
    IN-CAT - ICFCAT.VSTOR02
    HISTORY
      OWNER-IDENT---(NULL)      CREATION-----2002.129
      RELEASE---------2     EXPIRATION----2002.365
    VOLUMES
      VOLSER-------TSO001     DEVTYPE---X'3010200F'
```

Description

- The ALLOCATION listing is like the VOLUME listing with additional information about the extents used for each file.
- The ALL listing is like the ALLOCATION listing with additional information that includes attribute information.

How to use the ALTER and DELETE commands

The ALTER and DELETE commands are used to modify a catalog entry or to delete a data set. Since you will probably need to do both at one time or another, these commands are presented next.

How to use the ALTER command to modify a catalog entry

You can use the ALTER command to modify a catalog entry as shown in figure 16–12. The parameters in the syntax for this command let you change an entry's name or its volume allocations.

In the first example, this command is used to change an entry's name to MM01.CUSTMAST. This type of command can be used for a cluster, data component, index component, alternate index, path, or catalog.

In the second example, this command is used to remove two volumes that have been allocated for a data set and to add two volumes to the allocation. In this case, you need to use the data component as the entry to be altered, not the cluster. Note, however, that you can't remove a volume for a data set if space on that volume has already been allocated to the data set. To see what volumes have already been allocated, you can use the LISTCAT command with the ALLOCATION parameter.

Besides the parameters shown in this figure, you can use other DEFINE CLUSTER parameters to alter the catalog entry for a data set. For instance, the third example shows how the SHAREOPTIONS parameter can be used to change the share options for a data set. Note, however, that there are many restrictions on what parameters you can alter once a data set has been created. So you need to consult the appropriate AMS manual or check with a systems programmer at your installation if you absolutely have to change one of the parameters. The alternative is to redefine the data set with the changed parameters and create it anew from the current version of the data set.

Figure 16−12: How to use the ALTER command to modify a catalog entry

The syntax of the ALTER command

```
ALTER   entry-name
      [ CATALOG(name) ]
      [ NEWNAME(entry-name) ]
      [ ADDVOLUMES(vol-ser…) ]
      [ REMOVEVOLUMES(vol-ser…) ]
```

Explanation

entry−name	Specifies the name of the object whose catalog entry is to be altered.
CATALOG	Identifies the catalog that contains the object to be altered. Required only if the catalog can't be located by the standard search sequence.
NEWNAME	Specifies a new entry name for the entry.
ADDVOLUMES	Adds the specified volumes to the list of volumes where space may be allocated to the data component. Can't be used with a cluster.
REMOVEVOLUMES	Removes the specified volumes from the list of volumes where space may be allocated to the data component. Ignored if space has already been allocated on the specified volumes. Can't be used with a cluster.

A command that changes that name of a data set

```
ALTER MM01.CUSTOMER.MASTER    -
      NEWNAME(MM01.CUSTMAST)
```

A command that adds two volumes and removes two other volumes

```
ALTER MM01.CUSTOMER.MASTER.DATA    -
      ADDVOLUMES(VOL291 VOL292)    -
      REMOVEVOLUMES(VOL281 VOL282)
```

A command that changes the share options

```
ALTER MM01.CUSTOMER.MASTER.DATA  -
      SHAREOPTIONS(3 3)
```

Description

- You can use the ALTER command to change a VSAM object's name, volume allocation, and other characteristics. This command can be used for a cluster, data component, index component, alternate index, path, or catalog.
- Besides the parameters shown in the syntax above, you can code many of the DEFINE CLUSTER parameters like the FREESPACE or SHAREOPTIONS parameters. However, there are restrictions to the use of some of these, so be sure to consult your systems programmer or the appropriate AMS reference manual to see what these restrictions are.

How to use the DELETE command to delete a data set

You use the DELETE command to delete a data set as shown in figure 16–13. To identify the data sets that you want to delete, you can list their names as shown in the first two examples. Or, you can use a generic entry name or an entry type to identify the data sets to be deleted as shown in the third example. If you want to delete the data sets even if their expiration dates haven't yet been reached, you can code the PURGE parameter.

Normally, when a data set is deleted, its name is removed from the catalog, but the data set itself remains on the DASD. For most data sets, that's all that's needed because there's no way to access the data set once it's removed from the catalog. Eventually, then, it gets overwritten by other data sets. If that isn't enough for sensitive data sets, though, you can code the ERASE parameter so the data set itself is overwritten by binary zeros. But keep in mind that this is an unnecessary waste of computer time for most data sets.

Figure 16–13: How to use the DELETE command to delete a data set

The syntax of the DELETE command

```
DELETE   (entry-name…)
       [ CATALOG(name)   ]
       [ entry-type… ]
       [ PURGE | NOPURGE ]
       [ ERASE | NOERASE ]
```

Explanation

entry–name	Specifies the name of the entry or entries to be deleted. If you specify more than one entry name, you must enclose the list in parentheses.
CATALOG	Specifies the name of the catalog that owns the entries to be deleted. Required only if the correct catalog can't be found using the standard search sequence.
entry–type	Specifies that only entries of the listed types should be deleted. The valid entry types are the same as for the LISTCAT command.
PURGE | NOPURGE	PURGE means that an object should be deleted even if its retention period has not expired. NOPURGE means to delete entries only if their expiration dates have passed (the default).
ERASE | NOERASE	ERASE means that the data component of a cluster or alternate index should be erased (overwritten with binary zeros). NOERASE means that the data component should not be erased (the default).

A command that deletes a data set whether or not it has expired

```
DELETE MM01.CUSTOMER.MASTER -
       PURGE
```

A command that deletes three named data sets

```
DELETE (MM01.CUSTOMER.MASTER         -
        MM01.CUSTMAST.DISTRICT.AIX   -
        MM01.CUSTMAST.DISTRICT.PATH)
```

A command that deletes all alternate indexes for data sets that match a generic entry name

```
DELETE MM01.CUSTMAST.*.AIX  -
       ALTERNATEINDEX
```

Description

- You can use the DELETE command to remove the entries for one or more data sets from a VSAM catalog.
- You can use generic entry names to identify the data sets that you want removed.
- Although the DELETE command removes the catalog entry for a data set, the data set remains on the disk until it's overwritten by another data set.
- Although you can use the ERASE parameter to overwrite a data set when it is deleted, that can be time−consuming and it usually isn't necessary.

How to print and copy data sets

To print and copy data sets, you can use the AMS PRINT and REPRO commands. As you will see, these are useful commands that work for both VSAM and non−VSAM data sets.

How to use the PRINT command to print a data set

Figure 16−14 presents the syntax of the PRINT command with the parameters separated into four groups. You use one of the parameters in the first group to identify the data set to be printed. You can use the parameters in the second group to control how the printed output is handled. Then, if you don't want to print all of the records, you can use the parameters in the third and fourth groups to identify the records to be printed.

If you look at the examples in this figure, you can see how this works. In the first example, the INDATASET parameter identifies the data set to be printed, and CHARACTER identifies the printing format. Then, the SKIP parameter says to skip the first 28 records before starting and the COUNT parameter says to print just 3 records.

In the second example, a KSDS is printed in hex format starting with the record that has a key value of 1000 and ending with the record that has a key value of 1200. If the key in either parameter contains commas, semicolons, blanks, parentheses, or slashes, you must code the key between apostrophes, but normally that isn't necessary.

To give you some idea of what the printing formats look like, two lines of character and two lines of hex output are shown at the bottom of this figure. As you can see, the output for each record starts with its key. Although the character format works fine when each byte contains one character, numeric formats like packed−decimal are unreadable in this format. In that case, you need to use hex format, which presumes that you know how to decode the hex values. The dump format prints the data set in both character and hex format.

If you experiment with this command, you'll quickly see that it's easy to use. Remember too that it can be used with both VSAM and non−VSAM data sets, so it's a versatile command.

Figure 16−14: How to use the PRINT command to print a data set

The syntax of the PRINT command

```
PRINT { INDATASET(entry-name) }
      { INFILE(ddname)        }

      [ CHARACTER | HEX | DUMP ]
      [ OUTFILE(ddname) ]

      [ {SKIP(count)}
        {FROMKEY(key)}
        {FROMNUMBER(number)}
        {FROMADDRESS(address)} ]

      [ {COUNT(count)}
        {TOKEY(key)}
        {TONUMBER(number)}
        {TOADDRESS(address)} ]
```

Explanation

INDATASET	Specifies the name of the data set to be printed.
INFILE	Specifies the name of a DD statement that identifies the data set to be printed.
CHARACTER	Specifies that the data should be printed in character format.
HEX	Specifies that the data should be printed in hexadecimal format.
DUMP	Specifies that the data should be printed in both character and hex format.
OUTFILE	Specifies an output file other than the default SYSPRINT.
SKIP	Specifies the number of records to be skipped before the records are printed.
FROM…	Specifies the key (KSDS), number (RRDS), or relative−byte address (ESDS) of the first record to be printed.
COUNT	Specifies the number of records to be printed.
TO…	Specifies the key (KSDS), number (RRDS), or relative byte address (ESDS) of the last record to be printed.

A command that prints records 29, 30, and 31 in character format

```
PRINT INDATASET(MM01.CUSTOMER.MASTER)   -
      CHARACTER                         -
      SKIP(28)                          -
      COUNT(3)
```

A command that prints the records from key 1000 to key 1200 in hex format

```
PRINT INDATASET(MM01.CUSTOMER.MASTER)   -
      HEX                               -
      FROMKEY(1000)                     -
      TOKEY(1200)
```

The start of a listing in character format

```
KEY OF RECORD - 287760
287760JOHN WARDS AND ASSOC5600 N CLARKE         CHICAGO    IL603002027  0102
```

The start of the same listing in hex format

```
KEY OF RECORD - F2F8F7F7F6F0
F2F8F7F7F6F0D1D6C8D540E6C1D9C4E240C1D5C440C1E2E2D6C3F5F6F0F040D540C3D3C1D9D2
```

How to use the REPRO command to copy a data set

Figure 16−15 shows how to use the REPRO command to copy a data set. As you can see, its syntax is similar to the syntax of the PRINT command, but this time the parameters are separated into five groups.

To start, you use one of the parameters in the first group to identify the input data set and one of the parameters in the second group to identify the output data set. Then, if you don't want to copy all of the records in the input data set, you can use the parameters in the third and fourth groups to identify the records to be copied. This works just like it does in the PRINT command. In the example in this figure, the first 1000 records of the input data set are copied to the output data set.

You can use this command with both VSAM and non−VSAM data sets, and the input files can have different organizations. For instance, you can copy a non−VSAM sequential file into a VSAM KSDS. The only restriction is that you can't use a non−VSAM ISAM file as the output data set, but you shouldn't want to do that.

When you use this command, you must remember that the output data set must already exist. Then, if the output data set is empty, VSAM copies the input data set into the output data set. However, if the output data set contains records, VSAM merges the input records with the output records. For an ESDS, the input records are added at the end of the output data set. For a KSDS or RRDS, the records are added in their correct positions based on their key values or relative record numbers.

With that in mind, you can understand how the parameters in the last group work. If you code REUSE for a reusable data set, the input records overwrite the records that were in the output data set. If you code NOREUSE, the input records are merged with the existing records in the reusable data set.

Regardless of whether or not the data set is reusable, when the records are merged, the REPLACE and NOREPLACE parameters take effect. If you code REPLACE, records with duplicate keys or relative record numbers replace the corresponding records in the output data set. If you code NOREPLACE, the duplicate records in the output data set are retained and the duplicate input records are discarded.

Figure 16−15: How to use the REPRO command to copy a data set

The syntax of the REPRO command

```
REPRO { INDATASET(entry-name) }
      { INFILE(ddname)        }

      { OUTDATASET(entry-name) }
      { OUTFILE(ddname)        }

      [ {SKIP(count)}
        {FROMKEY(key)}
        {FROMNUMBER(number)}
        {FROMADDRESS(address)} ]

      [ {COUNT(count)}
        {TOKEY(key)}
        {TONUMBER(number)}
        {TOADDRESS(address)} ]

      [ REUSE | NOREUSE ]
      [ REPLACE | NOREPLACE ]
```

Explanation

INDATASET	Specifies the name of the data set to be copied.
INFILE	Specifies the name of a DD statement that identifies the data set to be copied.
OUTDATASET	Specifies the name of the output data set.

OUTFILE	Specifies the name of a DD statement that identifies the output data set.
SKIP	Specifies the number of records to be skipped before copying begins.
FROM…	Specifies the key (KSDS), number (RRDS), or relative byte address (ESDS) of the first record to be copied.
COUNT	Specifies the number of records to be copied.
TO…	Specifies the key (KSDS), number (RRDS), or relative byte address (ESDS) of the last record to be copied.
REUSE | <u>NOREUSE</u>	Specifies whether or not a reusable output file should be reset. If not, the new records are merged with the existing ones in the file.
REPLACE | <u>NOREPLACE</u>	Specifies whether or not duplicate records should be replaced.

A command that copies the first 1000 records in a data set

```
REPRO INDATASET(MMA2.CUSTOMER.MASTER)       -
      OUTDATASET(MMA2.CUSTOMER.MASTER.COPY) -
      COUNT(1000)
```

Description

- You can use the REPRO command to copy non–VSAM or VSAM data sets with any file organization, and the input and output files can have different organizations.

- The output data set in a REPRO command must exist when the job is run. If the output data set is empty, the records are copied from the input data set to the output data set. If the output data set contains records, the records from the input file are merged with the records in the output file depending on the data set's organization.

How to define and build an alternate index

To define and build an alternate index for a KSDS requires three different AMS commands. First, you use the DEFINE ALTERNATEINDEX command to define the alternate index. Next, you use the DEFINE PATH command to define a path that lets you process a base cluster via its alternate index. Then, you use the BLDINDEX command to create the alternate index from data that's derived from the base cluster.

How to use the DEFINE ALTERNATEINDEX command

Figure 16–16 gives the syntax of the DEFINE ALTERNATEINDEX (or AIX) command. As you can see, its syntax is similar to the syntax for the DEFINE CLUSTER command. That's because an alternate index is actually a KSDS. As a result, you should already know how to code most of the parameters for this command. Note, however, that there are also six new parameters that are summarized in this figure.

To name an alternate index, I suggest that you use a combination of the base cluster name, the name of the alternate key, and the letters AIX. I also suggest that you name the data and index components by adding DATA and INDEX to the name of the cluster. If, for example, social security number is going to be used as an alternate index for an employee master file, the index can be defined with names like these:

```
MM2.EMPMAST.SSN.AIX
MM2.EMPMAST.SSN.AIX.DATA
MM2.EMPMAST.SSN.AIX.INDEX
```

That way, you can easily identify the names when you list the catalog entries.

When you code the RELATE parameter, you supply the name of the base cluster that the index will apply to. Then, the KEYS parameter gives the length and offset of the alternate key in the base cluster. And the UNIQUEKEY and NONUNIQUEKEY parameters determine whether duplicate keys are allowed.

As for the UPGRADE and NOUPGRADE parameters, they determine whether an alternate index is part of the base cluster's *upgrade set*. If it is, the alternate index is updated whenever the base cluster is processed by primary key. Otherwise, the index must be upgraded as a separate function.

Because alternate indexes are frequently rebuilt as a separate function, I suggest that you define each alternate index with the REUSE parameter. Then, you can use the BLDINDEX command to rebuild the records of the alternate index without deleting and redefining the index.

Figure 16–16: The syntax of the DEFINE ALTERNATEINDEX command

The syntax of the DEFINE ALTERNATEINDEX command

```
DEFINE ALTERNATEINDEX (   NAME(entry-name)
                          RELATE(cluster-name)
                        [ OWNER(owner-id) ]
                        [ FOR(days) | TO(date) ]
                        [ KEYS(length offset) ]
                        [ UNIQUEKEY | NONUNIQUEKEY ]
                        [ UPGRADE | NOUPGRADE ]
                        [ VOLUMES(vol-ser…) ]
                        [ FILE(ddname) ]
                        [ {CYLINDERS(primary [secondary])}
                          {KILOBYTES(primary [secondary])}
                          {MEGABYTES(primary [secondary])}
                          {TRACKS(primary [secondary])   }
                          {RECORDS(primary [secondary])  } ]
                        [ REUSE | NOREUSE ]
                        [ SHAREOPTIONS(a b) ]
                        [ MODEL(entry-name [cat-name]) ] )
        [ DATA        ( [ NAME(entry-name) ]
                        [ VOLUMES(vol-ser…) ]
                        [ {CYLINDERS(primary [secondary])}
                          {KILOBYTES(primary [secondary])}
                          {MEGABYTES(primary [secondary])}
                          {TRACKS(primary [secondary])   }
                          {RECORDS(primary [secondary])  } ] ) ]
        [ INDEX       ( [ NAME(entry-name) ]
                        [ VOLUMES(vol-ser…) ]
                        [ {CYLINDERS(primary [secondary])}
                          {KILOBYTES(primary [secondary])}
                          {MEGABYTES(primary [secondary])}
                          {TRACKS(primary [secondary])   }
                          {RECORDS(primary [secondary])  } ] ) ]
        [ CATALOG(name) ]
```

Explanation of the parameters that aren't in the DEFINE CLUSTER command

RELATE	Specifies the name of the base cluster to which this alternate index applies.
UNIQUEKEY	Duplicate keys aren't allowed.
NONUNIQUEKEY	Duplicate keys are allowed.
UPGRADE	The alternate index is part of the base cluster's upgrade set.

NOUPGRADE	The alternate index isn't part of the base cluster's upgrade set.
MODEL	Specifies the name of an existing alternate index to use as a model.

Description

- Because an alternate index is actually a KSDS, the DEFINE ALTERNATEINDEX or DEFINE AIX command is similar to the DEFINE CLUSTER command.

How to use the DEFINE PATH command

To process the records of a base cluster via an alternate index, you need to define another catalog entry called a *path*. To do that, you use the DEFINE PATH command that's summarized in figure 16–17. Here, the first three parameters are the critical ones.

To start, you use the NAME parameter to provide a name for the path. As a naming convention, I suggest that you use the cluster name for the alternate index with PATH instead of AIX at the end of the name. Then, you use the RELATE parameter to supply the name of the alternate index. In other words, the DEFINE PATH command relates the path to the alternate index, and the DEFINE AIX command relates the alternate index to the base cluster.

The UPDATE or NOUPDATE parameter tells AMS whether or not you want the base cluster's upgrade set to be updated when you process the cluster via the path. If you specify UPDATE, all of the alternate indexes in the upgrade set are updated when you open the path. If you specify NOUPDATE, the upgrade set isn't updated when you process the file via the path.

The example in this figure shows how the DEFINE AIX and DEFINE PATH commands are related. Here, you can see the naming conventions used for the index and path. You can see that the alternate index has a unique key that's added to the upgrade set for the base cluster. And you can see that the DEFINE PATH command tells AMS to update the upgrade set when the base cluster is processed via the path.

You can also see that the alternate index is defined as a reusable data set. That way, the alternate index can be rebuilt whenever needed without having to be redefined.

Figure 16–17: How the DEFINE AIX and the DEFINE PATH commands work together

The syntax of the DEFINE PATH command

```
DEFINE PATH (   NAME(entry-name)
                PATHENTRY(aix-name)
              [ UPDATE | NOUPDATE ]
              [ FOR(days) | TO(date) ]
              [ MODEL(entry-name [cat-name]) ]
              [ RECATALOG | NORECATALOG ] )
 [ CATALOG(name) ]
```

Explanation

NAME	Specifies the name of the path.
PATHENTRY	Specifies the name of the alternate index to which this path is related.
UPDATE	

	The upgrade set should be updated when this path is processed.
NOUPDATE	The upgrade set shouldn't be updated when this path is processed.
MODEL	Specifies the name of an existing path to use as a model.
RECATALOG	Specifies that a path entry is to be recataloged. The NAME and PATHENTRY parameters must be specified as they were when the path was originally defined.
NORECATALOG	Specifies that a new path entry should be created in a catalog.
CATALOG	Specifies the name of the catalog that contains the alternate index. Required only if the correct catalog can't be found using the standard search sequence.

Two commands that define an alternate index and its path

```
DEFINE AIX    ( NAME(MMA2.EMPMAST.SSN.AIX)                -
                RELATE(MMA2.EMPMAST)                      -
                KEYS(9 12)                                -
                UNIQUEKEY                                 -
                UPGRADE                                   -
                REUSE                                     -
                VOLUMES(MPS800) )                         -
       DATA   ( NAME(MMA2.EMPMAST.SSN.AIX.DATA)           -
                CYLINDERS(1 1) )                          -
       INDEX  ( NAME(MMA2.EMPMAST.SSN.AIX.INDEX) )
DEFINE PATH   ( NAME(MMA2.EMPMAST.SSN.PATH)               -
                PATHENTRY(MMA2.EMPMAST.SSN.AIX)           -
                UPDATE )
```

Description

- A *path* lets you access base cluster records via an alternate index.
- You don't have to supply the name of the base cluster in the DEFINE PATH command because it is identified in the RELATE parameter of the DEFINE AIX command.
- A base cluster's *upgrade set* is all of its upgradeable alternate indexes. You add an alternate index to the upgrade set by coding the UPGRADE parameter in the DEFINE AIX command.
- The upgrade set is always updated when the base cluster is processed by its primary key. To update the upgrade set when the base cluster is processed by an alternate key (via a path), you code the UPDATE parameter in the DEFINE PATH command.

How to use the BLDINDEX command

Before you can process a base cluster via an alternate index, of course, you need to build the index. To do that, you use the BLDINDEX command that's summarized in figure 16–18. In this case, the base cluster is the input data set that you identify by one of the parameters in the first group, and the alternate index is the output data set that you identify by one of the parameters in the second group. However, you can use either the index name or the path name for the output data set.

When the BLDINDEX command is executed, it reads all the records in the base cluster and extracts the alternate key and primary key from each one. Next, it sorts these pairs of keys into ascending sequence by alternate key. Then, it writes the alternate index records to the alternate index. Along the way, if AMS finds duplicate keys for a UNIQUEKEY index, it flags the duplicates as errors.

Whenever possible, VSAM should do the sort as an internal sort because that's faster than an external sort. As a result, you usually shouldn't code the EXTERNALSORT parameter. If VSAM can't do an internal sort, though, it looks for two sort work files and does an external sort.

For that reason, you usually should provide DD statements for those work files. To do that, you can use the WORKFILES parameter to give the ddnames for the two work files. But it's easier to use the default ddnames (IDCUT1 and IDCUT2). In the example in this figure, you can see how the DD statements for those ddnames are coded.

Remember that you will not only use the BLDINDEX command to build an alternate index for the first time, but also to rebuild an index when it isn't part of the base cluster's upgrade set or when its path isn't defined for update. That's common practice because maintaining all of the alternate indexes as a KSDS is processed requires considerable overhead.

Figure 16–18: How to use the BLDINDEX command to build an alternate index

The syntax of the BLDINDEX command

```
BLDINDEX { INDATASET(cluster-name) }
         { INFILE(ddname)        }

         { OUTDATASET(aix-or-path-name) }
         { OUTFILE(ddname)        }

         [ EXTERNALSORT | INTERNALSORT ]

         [ WORKFILES(ddname ddname) ]

         [ CATALOG(name) ]
```

Explanation

INTERNALSORT	Specifies that the index records should be sorted in virtual storage. This is the default. Then, if VSAM can't find enough virtual storage for an internal sort, it automatically does an external sort.
EXTERNALSORT	Specifies that the index records should be sorted by an external sort program that uses disk work files. It's faster, though, to use the internal sort.
WORKFILES	Supplies the ddnames for the work files that are used by an external sort. If omitted, OS/390 uses IDCUT1 and IDCUT2 as the ddnames for these files, and you must provide DD statements for them.
CATALOG	Specifies the name of the catalog that will own the sort work files. If omitted, the standard search sequence is used.

A job that builds an alternate index

```
//MM01LC3  JOB  (36512),'R MENENDEZ',NOTIFY=&SYSUID
//         EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//IDCUT1   DD   UNIT=SYSDA,VOL=SER=MPS800,DISP=OLD
//IDCUT2   DD   UNIT=SYSDA,VOL=SER=MPS800,DISP=OLD
//SYSIN    DD   *
 BLDINDEX INDATASET(MMA2.EMPMAST)          -
          OUTDATASET(MMA2.EMPMAST.SSN.AIX)  -
          CATALOG(MMA2)
/*
```

Description

- You use the BLDINDEX command to build an alternate index for the first time. You also use it to rebuild the index whenever that’s necessary.
- To build an alternate index, VSAM first reads all the records in the base cluster and extracts the primary and alternate keys for each record. Then, it sorts the key pairs into ascending sequence by alternate key and writes the alternate index to its KSDS. If an alternate index allows nonunique keys, VSAM combines the duplicate primary keys for each alternate key into a single alternate index record.
- Unless you code the EXTERNALSORT parameter, VSAM tries to do the sort internally. If it can’t, it does an external sort using two work files that you must provide DD statements for.

Perspective

The goal of this chapter has been to show you how to use the AMS commands that most application programmers need. If this chapter has succeeded, you should now know what these commands can do. Then, when you need to use one of them later on, you can refer back to this chapter for the specific coding details.

Keep in mind, though, that there’s more to AMS than this chapter shows. In particular, there are other parameters and considerations for improving the performance of production data sets. Most shops, however, have a systems programmer or VSAM expert who specializes in fine–tuning the performance of VSAM data sets when they are put into production. As a result, the application programmer doesn’t need that level of VSAM mastery.

Terms

functional command

modal command

IDCAMS

retention period

expiration date

cross–region share option

cross–system share option

generic entry name

upgrade set

path