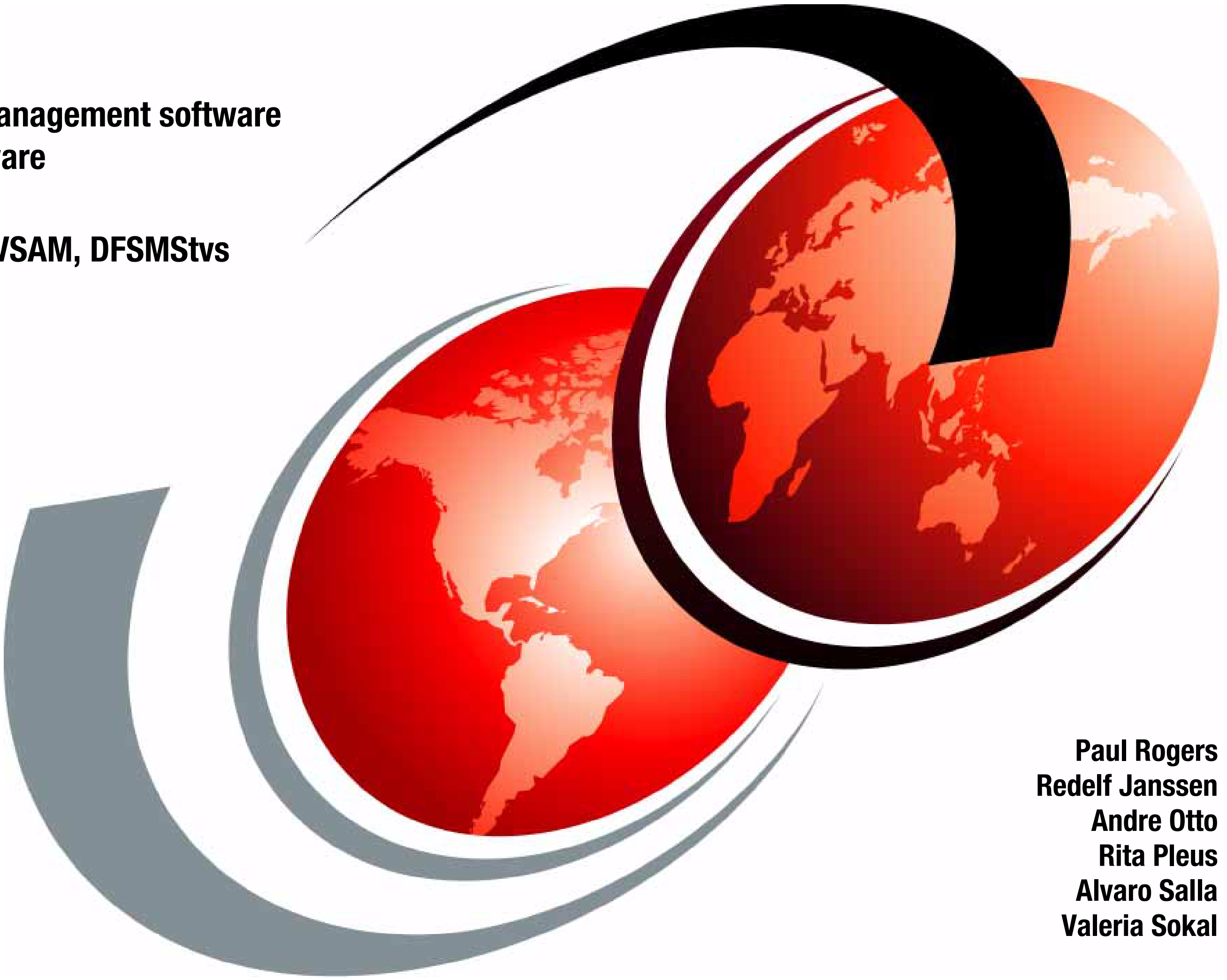
IBM

ABCs of z/OS System Programming Volume 3

DFSMS, Data set basics, SMS

Storage management software and hardware

Catalogs, VSAM, DFSMStvs



Paul Rogers
Redelf Janssen
Andre Otto
Rita Pleus
Alvaro Salla
Valeria Sokal

Redbooks

ibm.com/redbooks

International Technical Support Organization

ABCs of z/OS System Programming Volume 3

August 2007

SG24-6983-02

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Third Edition (August 2007)

This edition applies to Version 1 Release 8 of z/OS (5694-A01), Version 1 Release 8 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Notices

Trademarks

Preface

The ABCs of z/OS® System Programming is an eleven volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects. If you would like to become more familiar with z/OS in your current environment, or if you are evaluating platforms to consolidate your e-business applications, the ABCs collection will serve as a powerful technical tool.

The contents of the volumes are:

Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, and z/OS delivery and installation

Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, LNKLST, authorized libraries, Language Environment®, and SMP/E

Volume 3: Introduction to DFSMS™, data set basics, storage management hardware and software, VSAM, System-managed storage, catalogs, and DFSMStvs

Volume 4: Communication Server, TCP/IP and VTAM®

Volume 5: Base and Parallel Sysplex®, System Logger, Resource Recovery Services (RRS), global resource serialization (GRS), z/OS system operations, automatic restart management (ARM), Geographically dispersed Parallel Sysplex (GPDS)

Volume 6: Introduction to security, RACF®, Digital certificates and PKI, Kerberos, cryptography and z990 integrated cryptography, zSeries® firewall technologies, LDAP, Enterprise identity mapping (EIM), and firewall technologies

Volume 7: Printing in a z/OS environment, Infoprint® Server and Infoprint Central

Volume 8: An introduction to z/OS problem diagnosis

Volume 9: z/OS UNIX® System Services

Volume 10: Introduction to z/Architecture®, zSeries processor design, zSeries connectivity, LPAR concepts, HCD, and HMC

Volume 11: Capacity planning, performance management, WLM, RMF™, and SMF

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Paul Rogers is a Consulting IT Specialist at the International Technical Support Organization, Poughkeepsie Center and has worked for IBM® for more than 40 years. He writes extensively and teaches IBM classes worldwide on various aspects of z/OS, JES3, Infoprint Server, and z/OS UNIX. Before joining the ITSO 20 years ago, Paul worked in the

IBM Installation Support Center (ISC) in Greenford, England, providing OS/390® and JES support for IBM EMEA and the Washington Systems Center in Gaithersburg, Maryland.

Redelf Janssen is an IT Architect in IBM Global Services ITS in IBM Germany. He holds a degree in Computer Science from University of Bremen and joined IBM Germany in 1988. His areas of expertise include IBM zSeries, z/OS and availability management. He has written IBM Redbooks® publications on OS/390 Releases 3, 4, and 10, and z/OS Release 8.

Andre Otto is a z/OS DFSMS SW service specialist at the EMEA Backoffice team in Germany. He has 12 years of experience in the DFSMS, VSAM and catalog components. He holds a degree in Computer Science from the Dresden Professional Academy.

Rita Pleus is an IT Architect in IBM Global Services ITS in IBM Germany. She has 21 years of IT experience in a variety of areas, including systems programming and operations management. Before joining IBM in 2001, she worked for a German S/390® customer. Rita holds a degree in Computer Science from the University of Applied Sciences in Dortmund. Her areas of expertise include z/OS, its subsystems, and systems management.

Alvaro Salla is an IBM retiree who worked for IBM for more than 30 years in large systems. He has co-authored many IBM Redbooks publications and spent many years teaching S/360™ to S/390. He has a degree in Chemical Engineering from the University of Sao Paulo, Brazil.

Valeria Sokal is an MVS™ system programmer at an IBM customer. She has 16 years of experience as a mainframe systems programmer.

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

DFSMS introduction

This chapter gives a brief overview of the Data Facility Storage Management Subsystem (DFSMS) and its primary functions in the z/OS operating system. DFSMS comprises a suite of related data and storage management products for the z/OS system. DFSMS is now an integral element of the z/OS operating system.

DFSMS is an operating environment that helps automate and centralize the management of storage based on the policies that your installation defines for availability, performance, space, and security.

The heart of DFSMS is the Storage Management Subsystem (SMS). Using SMS, the storage administrator defines policies that automate the management of storage and hardware devices. These policies describe data allocation characteristics, performance and availability goals, backup and retention requirements, and storage requirements for the system.

DFSMS is an exclusive element of the z/OS operating system and is a software suite that automatically manages data from creation to expiration.

DFSMSdfp™ is a base element of z/OS. DFSMSdfp is automatically included with z/OS. DFSMSdfp performs the essential data, storage, and device management functions of the system. DFSMSdfp and DFSMShsm™ provide disaster recovery functions such as Advanced Copy Services and aggregate backup and recovery support (ABARS).

The other elements of DFSMS—DFSMSdss™, DFSMShsm, DFSMSrmm™, and DFSMStvs—are optional features that complement DFSMSdfp to provide a fully integrated approach to data and storage management. In a system-managed storage environment, DFSMS automates and centralizes storage management based on the policies that your installation defines for availability, performance, space, and security. With the optional features enabled, you can take full advantage of all the functions that DFSMS offers.

1.1 Introduction to DFSMS

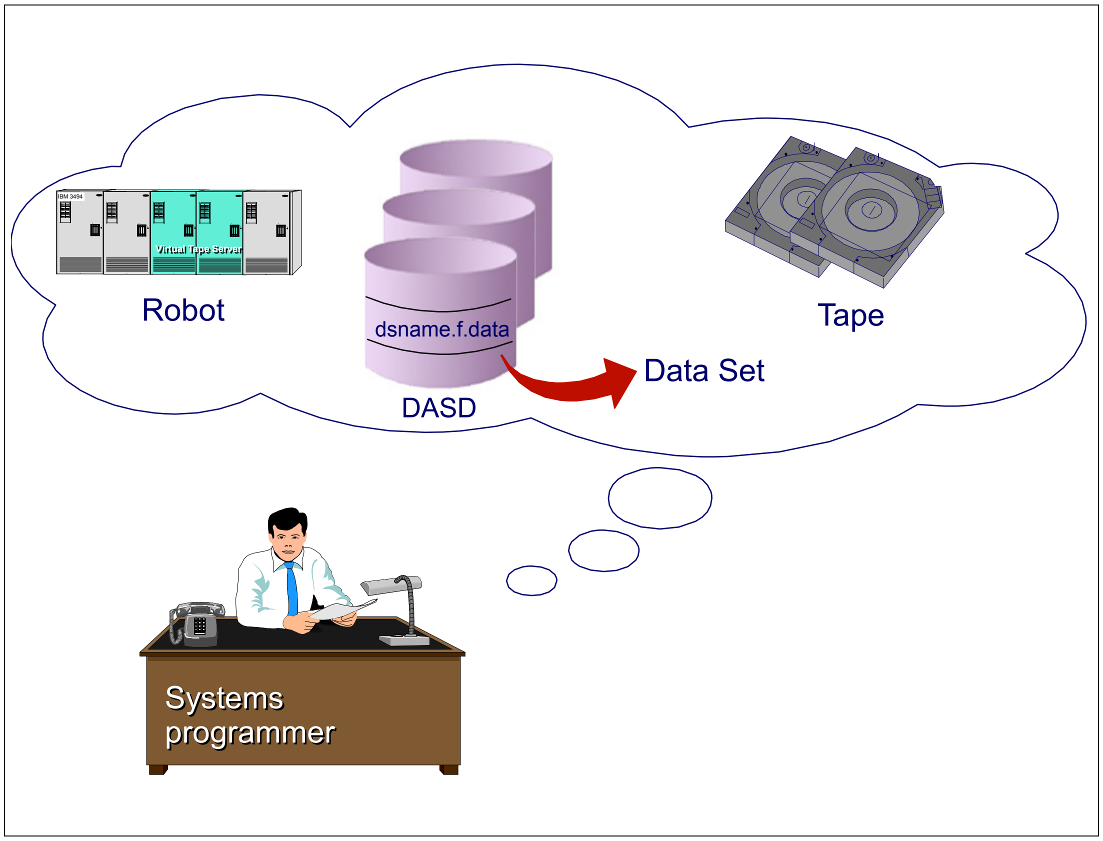


Figure 1-1 Introduction to data management

Understanding DFSMS

Data management is the part of the operating system that organizes, identifies, stores, catalogs, and retrieves all the data information (including programs) that your installation uses.

DFSMSdfp helps you store and catalog information on DASD, optical, and tape devices so that it can be quickly identified and retrieved from the system. DFSMSdfp provides access to both record- and stream-oriented data in the z/OS environment.

Systems programmer

As a systems programmer, you can use DFSMS data management to:

- Allocate space on DASD and optical volumes
- Automatically locate cataloged data sets
- Control access to data
- Transfer data between the application program and the medium
- Mount magnetic tape volumes in the drive

1.2 Data Facility Storage Management Subsystem

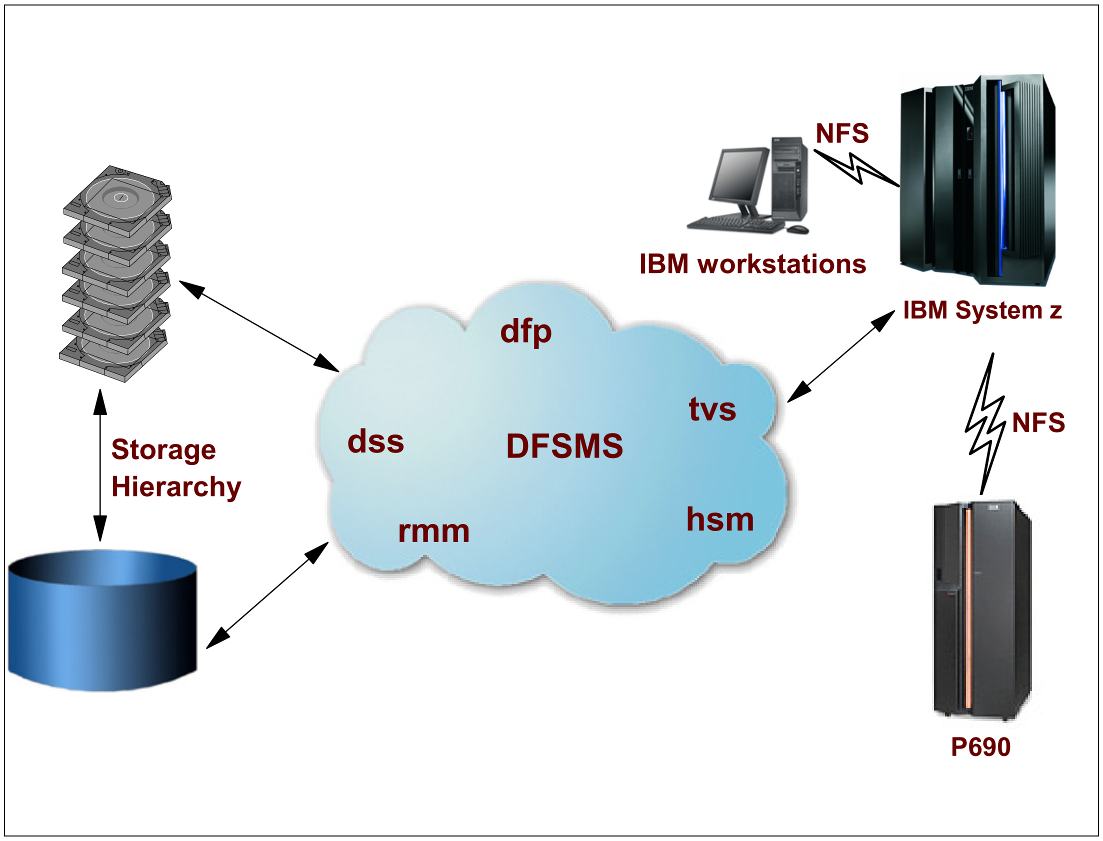


Figure 1-2 Data Facility Storage Management Subsystem

DFSMS components

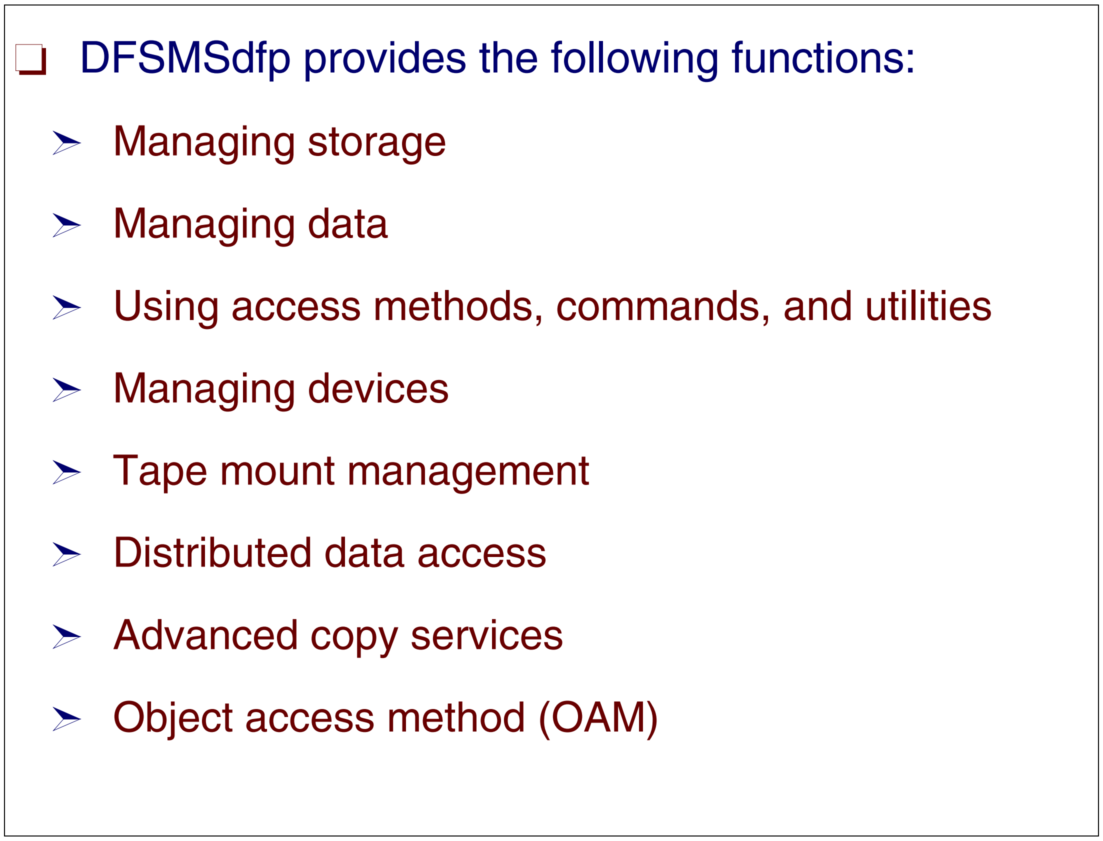
DFSMS is a set of products associated with z/OS that is responsible for data management. DFSMS has five MVS data management functional components as an integrated single software package:

DFSMSdfp	Provides storage, data, program, and device management. It is comprised of components such as access methods, OPEN/CLOSE/EOV routines, catalog management, DADSM (DASD space control), utilities, IDCAMS, SMS, NFS, ISMF, and other functions.
DFSMSdss	Provides data movement, copy, backup, and space management functions.
DFSMShsm	Provides backup, recovery, migration, and space management functions. It invokes DFSMSdss for certain of its functions.
DFSMSrmm	Provides management functions for removable media such as tape cartridges and optical media.
DFSMStvs	Enables batch jobs and CICS® online transactions to update shared VSAM data sets concurrently.

Network File System

The Network File System (NFS) is a distributed file system that enables users to access UNIX files and directories that are located on remote computers as though they were local. NFS is independent of machine types, operating systems, and network architectures.

1.3 DFSMSdfp component



❑ DFSMSdfp provides the following functions:

- Managing storage
- Managing data
- Using access methods, commands, and utilities
- Managing devices
- Tape mount management
- Distributed data access
- Advanced copy services
- Object access method (OAM)

Figure 1-3 DFSMSdfp functions

DFSMSdfp component

DFSMSdfp provides storage, data, program, and device management. It is comprised of components such as access methods, OPEN/CLOSE/EOV routines, catalog management, DADSM (DASD space control), utilities, IDCAMS, SMS, NFS, ISMF, and other functions.

Managing storage

The storage management subsystem (SMS) is a DFSMSdfp facility designed for automating and centralizing storage management. SMS automatically assigns attributes to new data when that data is created. SMS automatically controls system storage and assigns data to the appropriate storage device. ISMF panels allow you to specify these data attributes.

For more information about ISMF, see “Interactive Storage Management Facility (ISMF)” on page 289.

Managing data

DFSMSdfp organizes, identifies, stores, catalogs, shares, and retrieves all the data that your installation uses. You can store data on DASD, magnetic tape volumes, or optical volumes. Using data management, you can complete the following tasks:

- Allocate space on DASD and optical volumes
- Automatically locate cataloged data sets
- Control access to data
- Transfer data between the application program and the medium
- Mount magnetic tape volumes in the drive

Using access methods, commands, and utilities

DFSMSdfp manages the organization and storage of data in the z/OS environment. You can use access methods with macro instructions to organize and process a data set or object. Access method services commands manage data sets, volumes, and catalogs. Utilities perform tasks such as copying and moving data. You can use system commands to display and set SMS configuration parameters, use DFSMSdfp callable services to write advanced application programs, and use installation exits to customize DFSMS.

Managing devices with DFSMSdfp

You need to use the Hardware Configuration Definition (HCD) to define I/O devices to the operating system, and to control these devices. DFSMS manages DASD, storage control units, magnetic tape devices, optical devices, and printers. You can use DFSMS functions to manage many different device types, but most functions apply specifically to one type or one family of devices.

Tape mount management

Tape mount management is a methodology for improving tape usage and reducing tape costs. This methodology involves intercepting selected tape data set allocations through the SMS automatic class selection (ACS) routines and redirecting them to a direct access storage device (DASD) buffer. Once on DASD, you can migrate these data sets to a single tape or small set of tapes, thereby reducing the overhead associated with multiple tape mounts.

Distributed data access with DFSMSdfp

In the distributed computing environment, applications must often access data residing on different computers in a network. Often, the most effective data access services occur when applications can access remote data as though it were local data.

Distributed FileManager/MVS is a DFSMSdfp client/server product that enables remote clients in a network to access data on z/OS systems. Distributed FileManager/MVS provides workstations with access to z/OS data. Users and applications on heterogeneous client computers in your network can take advantage of system-managed storage on z/OS, data sharing, and data security with RACF.

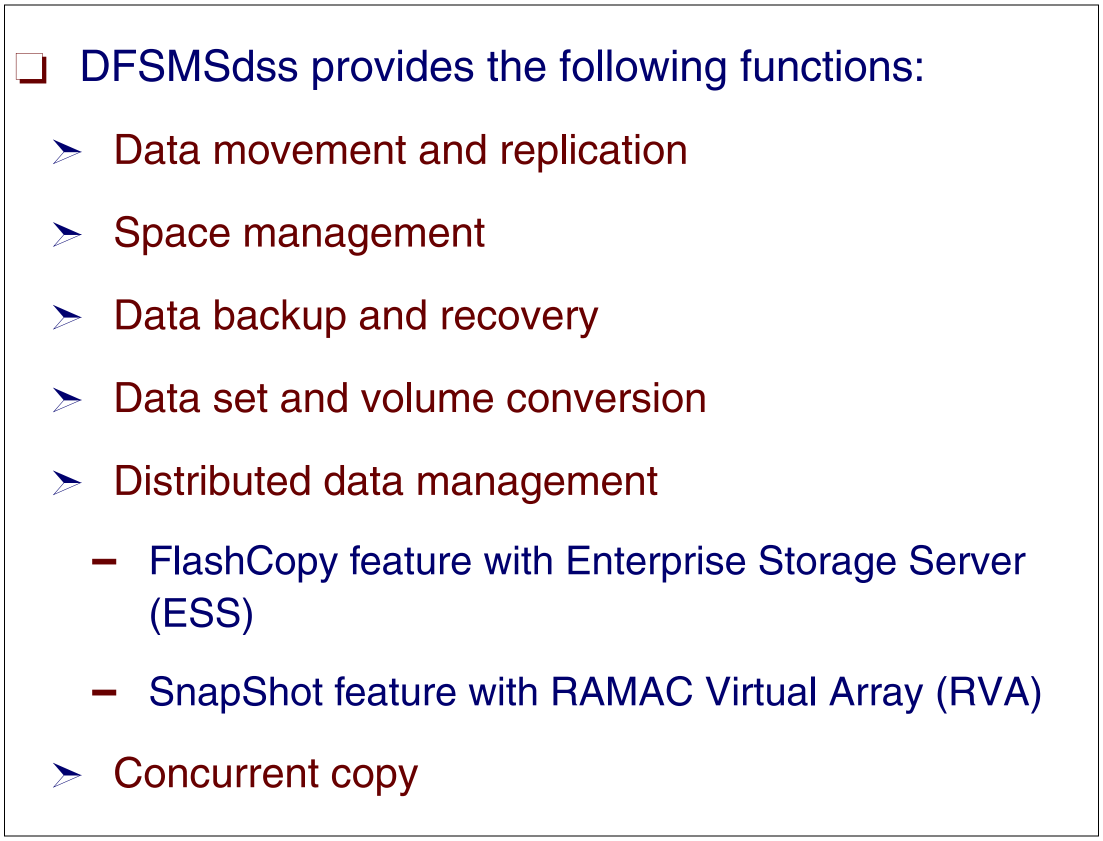
Advanced Copy Services

Advanced Copy Services includes remote and point-in-time copy functions that provide backup and recovery of data. When used before a disaster occurs, Advanced Copy Services provides rapid backup of critical data with minimal impact to business applications. If a disaster occurs to your data center, Advanced Copy Services provides rapid recovery of critical data.

Object access method

Object access method (OAM) provides storage, retrieval, and storage hierarchy management for objects. OAM also manages storage and retrieval for tape volumes that are contained in system-managed libraries.

1.4 DFSMSdss component



- DFSMSdss provides the following functions:
  - Data movement and replication
  - Space management
  - Data backup and recovery
  - Data set and volume conversion
  - Distributed data management
    - FlashCopy feature with Enterprise Storage Server (ESS)
    - SnapShot feature with RAMAC Virtual Array (RVA)
  - Concurrent copy

Figure 1-4 DFSMSdss functions

DFSMSdss component

DFSMSdss is the primary data mover for DFSMS. DFSMSdss copies and moves data to help manage storage, data, and space more efficiently. It can efficiently move multiple data sets from old to new DASD. The data movement capability that is provided by DFSMSdss is useful for many other operations, as well. You can use DFSMSdss to perform the following tasks.

Data movement and replication

DFSMSdss lets you move or copy data between volumes of like and unlike device types. If you create a backup in DFSMSdss, you can copy a backup copy of data. DFSMSdss also can produce multiple backup copies during a dump operation.

Space management

DFSMSdss can reduce or eliminate DASD free-space fragmentation.

Data backup and recovery

DFSMSdss provides you with host system backup and recovery functions at both the data set and volume levels. It also includes a stand-alone restore program that you can run without a host operating system.

Data set and volume conversion

DFSMSdss can convert your data sets and volumes to system-managed storage. It can also return your data to a non-system-managed state as part of a recovery procedure.

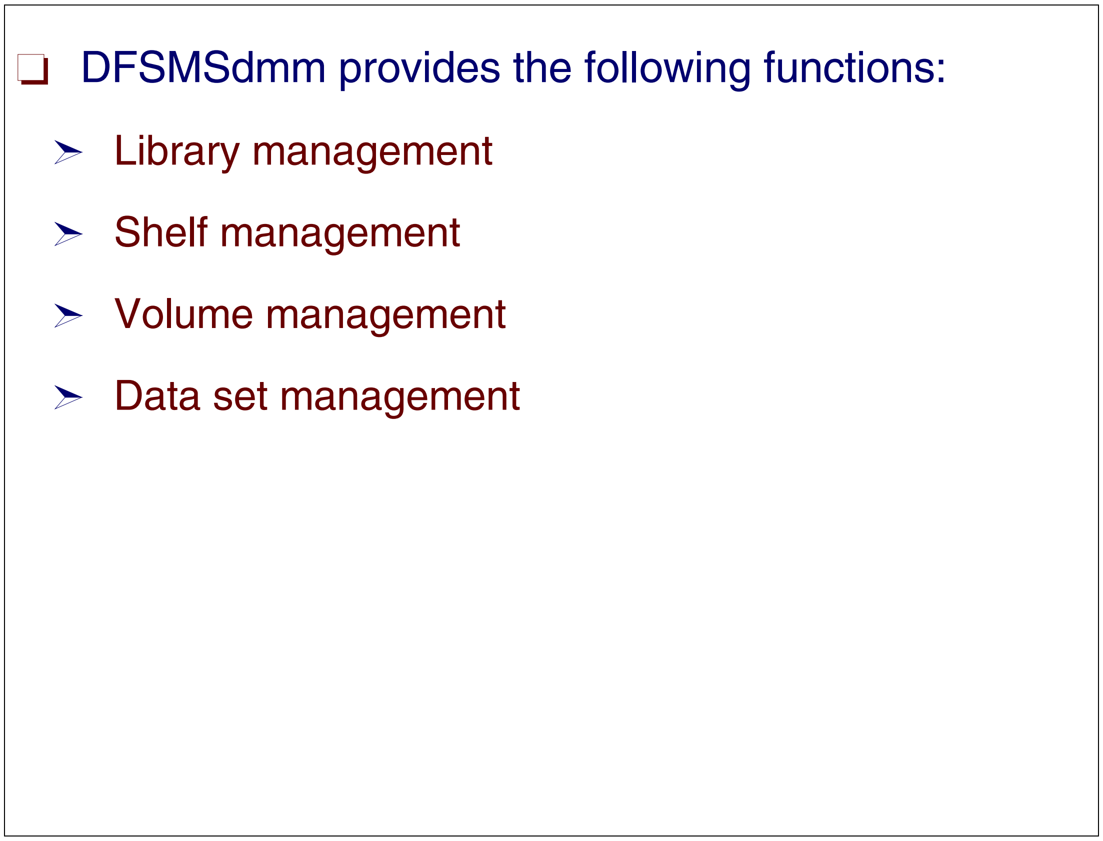
Distributed data management

DFSMSdss saves distributed data management (DDM) attributes that are associated with a specific data set and preserves those attributes during copy and move operations. DFSMSdss also offers the FlashCopy® feature with Enterprise Storage Server® (ESS) and the SnapShot feature with RAMAC® Virtual Array (RVA). FlashCopy and SnapShot function automatically, work much faster than traditional data movement methods, and are well-suited for handling large amounts of data.

Concurrent copy

When it is used with supporting hardware, DFSMSdss also provides concurrent copy capability. Concurrent copy lets you copy or back up data while that data is being used. The user or application program determines when to start the processing, and the data is copied as if no updates have occurred.

1.5 DFSMSrmm component



❑ DFSMSdmm provides the following functions:

➢ Library management

➢ Shelf management

➢ Volume management

➢ Data set management

Figure 1-5 DFSMSrmm functions

DFSMSrmm component

DFSMSrmm manages your removable media resources, including tape cartridges and reels. It provides the following functions.

Library management

You can create tape libraries, or collections of tape media associated with tape drives, to balance the work of your tape drives and help the operators that use them.

DFSMSrmm can manage the following devices:

- A removable media library, which incorporates all other libraries, such as:
  - System-managed manual tape libraries.
  - System-managed automated tape libraries. Examples of automated tape libraries include:
    - IBM TotalStorage®
    - Enterprise Automated Tape Library (3494)
    - IBM TotalStorage
    - Virtual Tape Servers (VTS)
- Non-system-managed or traditional tape libraries, including automated libraries such as a library under Basic Tape Library Support (BTLS) control.

Shelf management

DFSMSrmm groups information about removable media by shelves into a central online inventory, and keeps track of the volumes residing on those shelves. DFSMSrmm can manage the shelf space that you define in your removable media library and in your storage locations.

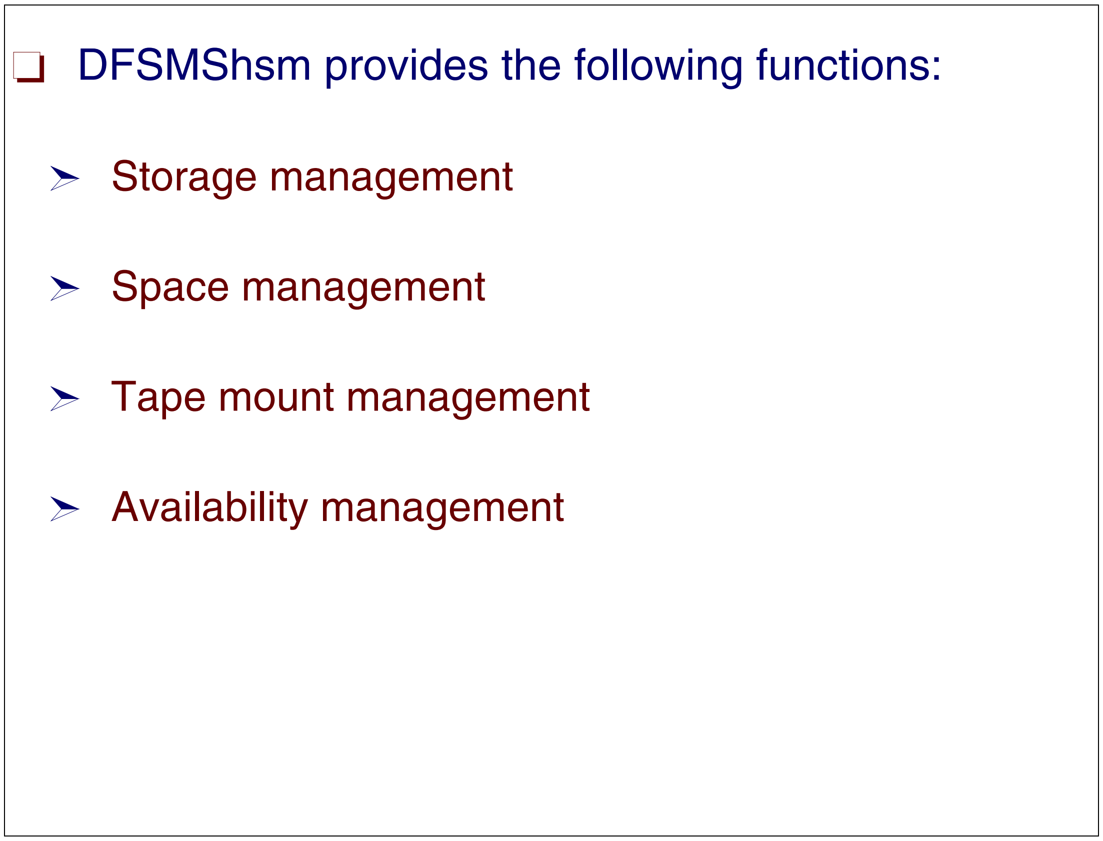
Volume management

DFSMSrmm manages the movement and retention of tape volumes throughout their life cycle.

Data set management

DFSMSrmm records information about the data sets on tape volumes. DFSMSrmm uses the data set information to validate volumes and to control the retention and movement of those data sets.

1.6 DFSMShsm component



- DFSMShsm provides the following functions:
  - Storage management
  - Space management
  - Tape mount management
  - Availability management

Figure 1-6 DFSMShsm functions

DFSMShsm component

DFSMShsm complements DFSMSdss to provide the following functions.

Storage management

DFSMShsm provides automatic DASD storage management, thus relieving users from manual storage management tasks.

Space management

DFSMShsm improves DASD space usage by keeping only active data on fast-access storage devices. It automatically frees space on user volumes by deleting eligible data sets, releasing overallocated space, and moving low-activity data to lower cost-per-byte devices, even if the job did not request tape.

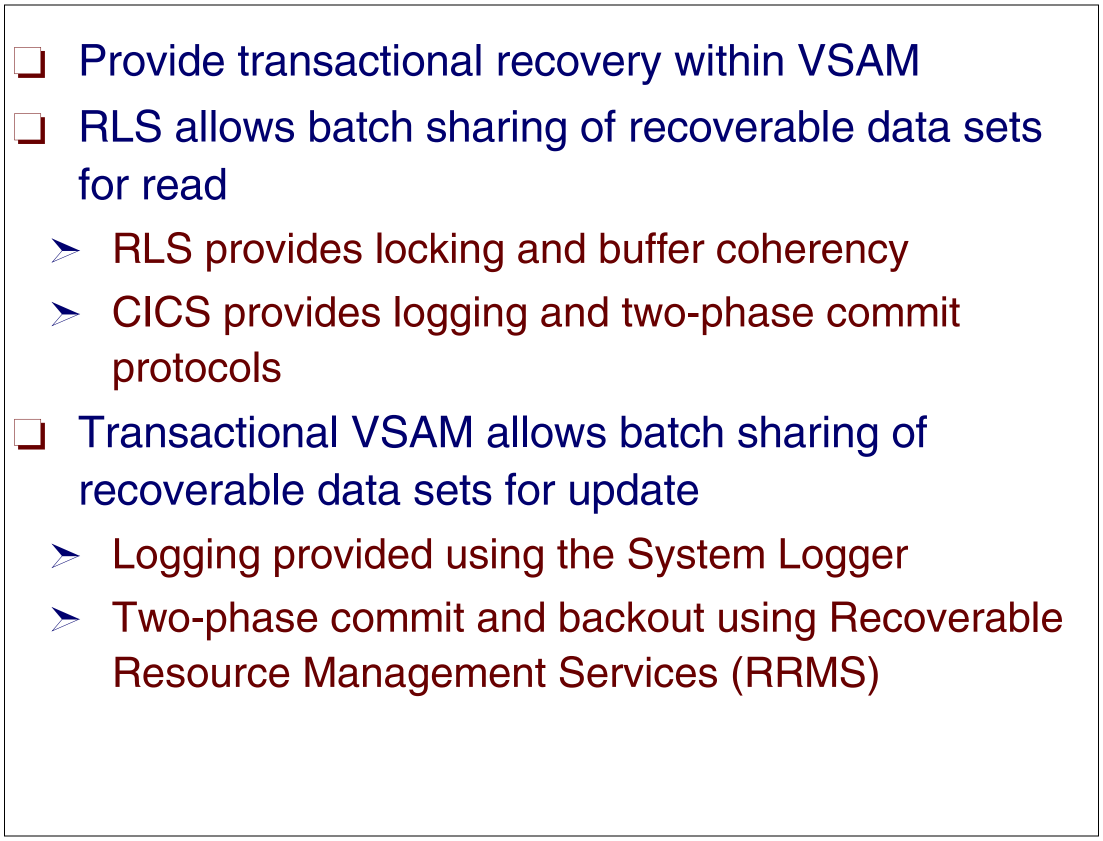
Tape mount management

DFSMShsm can write multiple output data sets to a single tape, making it a useful tool for implementing tape mount management under SMS. When you redirect tape data set allocations to DASD, DFSMShsm can move those data sets to tape, as a group, during interval migration. This methodology greatly reduces the number of tape mounts on the system. DFSMShsm uses a single-file format, which improves your tape usage and search capabilities.

Availability management

DFSMShsm backs up your data—automatically or by command—to ensure availability if accidental loss of the data sets or physical loss of volumes should occur. DFSMShsm also allows the storage administrator to copy backup and migration tapes, and to specify that copies be made in parallel with the original. You can store the copies onsite as protection from media damage, or offsite as protection from site damage. DFSMShsm also provides disaster backup and recovery for user-defined groups of data sets (aggregates) so that you can restore critical applications at the same location or at an offsite location.

1.7 DFSMStvs component



- ❑ Provide transactional recovery within VSAM
- ❑ RLS allows batch sharing of recoverable data sets for read
  - ➢ RLS provides locking and buffer coherency
  - ➢ CICS provides logging and two-phase commit protocols
- ❑ Transactional VSAM allows batch sharing of recoverable data sets for update
  - ➢ Logging provided using the System Logger
  - ➢ Two-phase commit and backout using Recoverable Resource Management Services (RRMS)

Figure 1-7 DFSMStvs functions

DFSMStvs component

DFSMS Transactional VSAM Services (DFSMStvs) allows you to share VSAM data sets across CICS, batch, and object-oriented applications on z/OS or distributed systems. DFSMStvs enables concurrent shared updates of recoverable VSAM data sets by CICS transactions and multiple batch applications. DFSMStvs enables 24-hour availability of CICS and batch applications.

VSAM record-level sharing (RLS)

With VSAM RLS, multiple CICS systems can directly access a shared VSAM data set, eliminating the need to ship functions between the application-owning regions and file-owning regions. CICS provides the logging, commit, and backout functions for VSAM recoverable data sets. VSAM RLS provides record-level serialization and cross-system caching. CICSVR provides a forward recovery utility.

DFSMStvs is built on top of VSAM record-level sharing (RLS), which permits sharing of recoverable VSAM data sets at the record level. Different applications often need to share VSAM data sets. Sometimes the applications need only to read the data set. Sometimes an application needs to update a data set while other applications are reading it. The most complex case of sharing a VSAM data set is when multiple applications need to update the data set and all require complete data integrity.

Transaction processing provides functions that coordinate work flow and the processing of individual tasks for the same data sets. VSAM record-level sharing and DFSMStvs provide key functions that enable multiple batch update jobs to run concurrently with CICS access to the same data sets, while maintaining integrity and recoverability.

Recoverable resource management services (RRMS)

RRMS is part of the operating system and comprises registration services, context services, and recoverable resource services (RRS). RRMS provides the context and unit of recovery management under which DFSMStvs participates as a recoverable resource manager.

2

Data set basics

A *data set* is a collection of logically related data; it can be a source program, a library of macros, or a file of data records used by a processing program. Data records (also called logical records) are the basic unit of information used by a processing program. By placing your data into volumes of organized data sets, you can save and process the data efficiently. You can also print the contents of a data set, or display the contents on a terminal.

You can store data on secondary storage devices, such as:

- A direct access storage device (DASD)

  The term DASD applies to disks or to a large amount of magnetic storage media on which a computer stores data. A *volume* is a standard unit of secondary storage. You can store all types of data sets on DASD.

  Each block of data on a DASD volume has a distinct location and a unique address, thus making it possible to find any record without extensive searching. You can store and retrieve records either directly or sequentially. Use DASD volumes for storing data and executable programs, including the operating system itself, and for temporary working storage. You can use one DASD volume for many different data sets, and reallocate or reuse space on the volume.

- A magnetic tape volume

  Only sequential data sets can be stored on magnetic tape. Mountable tape volumes can reside in an automated tape library. For information about magnetic tape volumes, see *z/OS DFSMS: Using Magnetic Tapes*, SC26-7412. You can also direct a sequential data set to or from spool, a UNIX file, a TSO/E terminal, a unit record device, virtual I/O (VIO), or a dummy data set.

The Storage Management Subsystem (SMS) is an operating environment that automates the management of storage. Storage management uses the values provided at allocation time to determine, for example, on which volume to place your data set, and how many tracks to allocate for it. Storage management also manages tape data sets on mountable volumes that reside in an automated tape library. With SMS, users can allocate data sets more easily.

The data sets allocated through SMS are called *system-managed data sets* or *SMS-managed data sets*.

An *access method* is a DFSMSdfp component that defines the technique that is used to store and retrieve data. Access methods have their own data set structures to organize data, macros to define and process data sets, and utility programs to process data sets.

Access methods are identified primarily by the way that they organize the data in the data set. For example, use the basic sequential access method (BSAM) or queued sequential access method (QSAM) with sequential data sets. However, there are times when an access method identified with one organization can be used to process a data set organized in a different manner. For example, a sequential data set (not extended-format data set) created using BSAM can be processed by the basic direct access method (BDAM), and vice versa. Another example is UNIX files, which you can process using BSAM, QSAM, basic partitioned access method (BPAM), or virtual storage access method (VSAM).

This chapter describes some data set basics:

- Data set name rules
- Data set characteristics
- Locating a data set
- Volume table of contents (VTOC)
- Initializing a volume

2.1 Data sets on storage devices

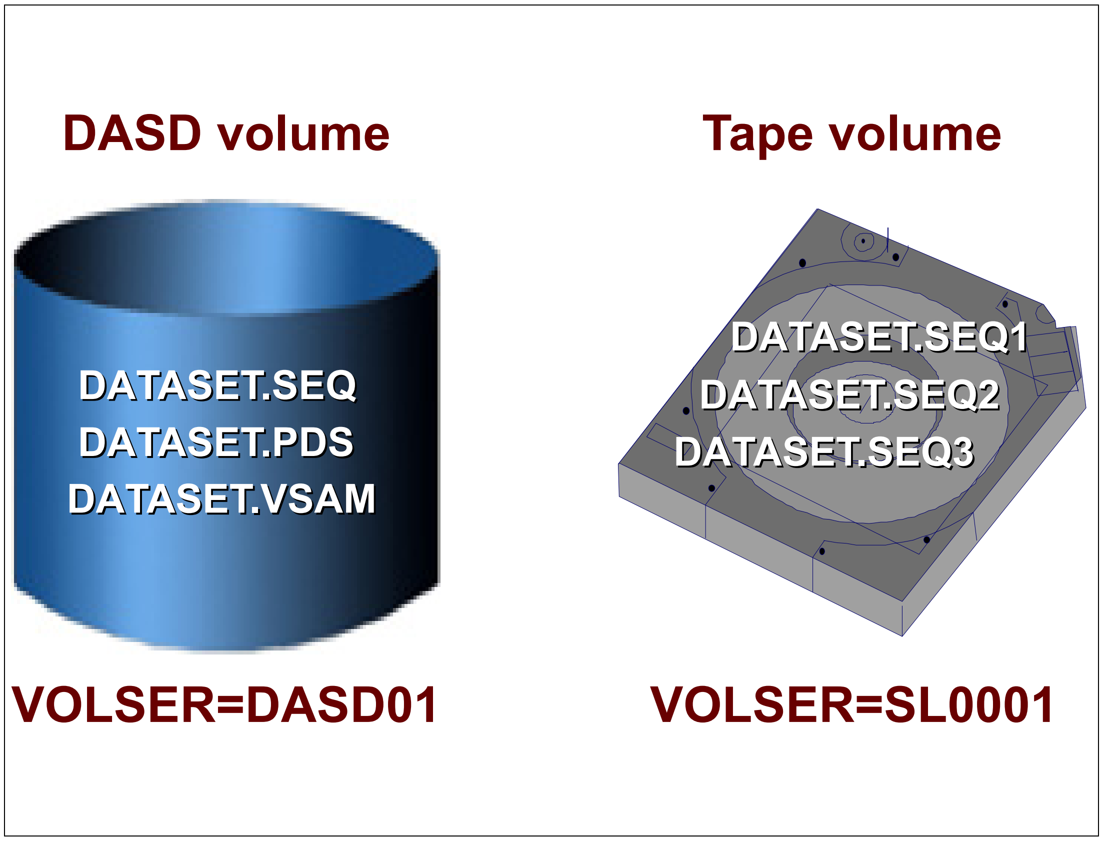


Figure 2-1 Data sets on volumes

MVS data sets

An *MVS data set* is a collection of logically related data records stored on one volume or a set of volumes. A data set can be, for example, a source program, a library of macros, or a file of data records used by a processing program. You can print a data set or display it on a terminal. The *logical record* is the basic unit of information used by a processing program.

Note: As an exception, the z/OS UNIX services component supports Hierarchical File System (HFS) data sets, where the collection is of bytes and there is not the concept of logically related data records.

Storage devices

Data can be stored on a magnetic direct access storage device (DASD), magnetic tape volume, or optical media. As mentioned previously, the term DASD applies to disks or simulated equivalents of disks. All types of data sets can be stored on DASD, but only sequential data sets can be stored on magnetic tape. The types of data sets are described in "DFSMSdfp data set types" on page 20.

DASD volumes

Each block of data on a DASD volume has a distinct location and a unique address, making it possible to find any record without extensive searching. You can store and retrieve records either directly or sequentially. Use DASD volumes for storing data and executable programs, including the operating system itself, and for temporary working storage. You can use one DASD volume for many different data sets, and reallocate or reuse space on the volume.

The following sections discuss the logical attributes of a data set, which are specified at data set creation time in:

- DCB/ACB control blocks in the application program
- DD cards (explicitly, or through the Data Class (DC) option with DFSMS)
- In an ACS Data Class (DC) routine (overridden by a DD card)

After the creation, such attributes are kept in catalogs and VTOCs.

2.2 Data set name rules

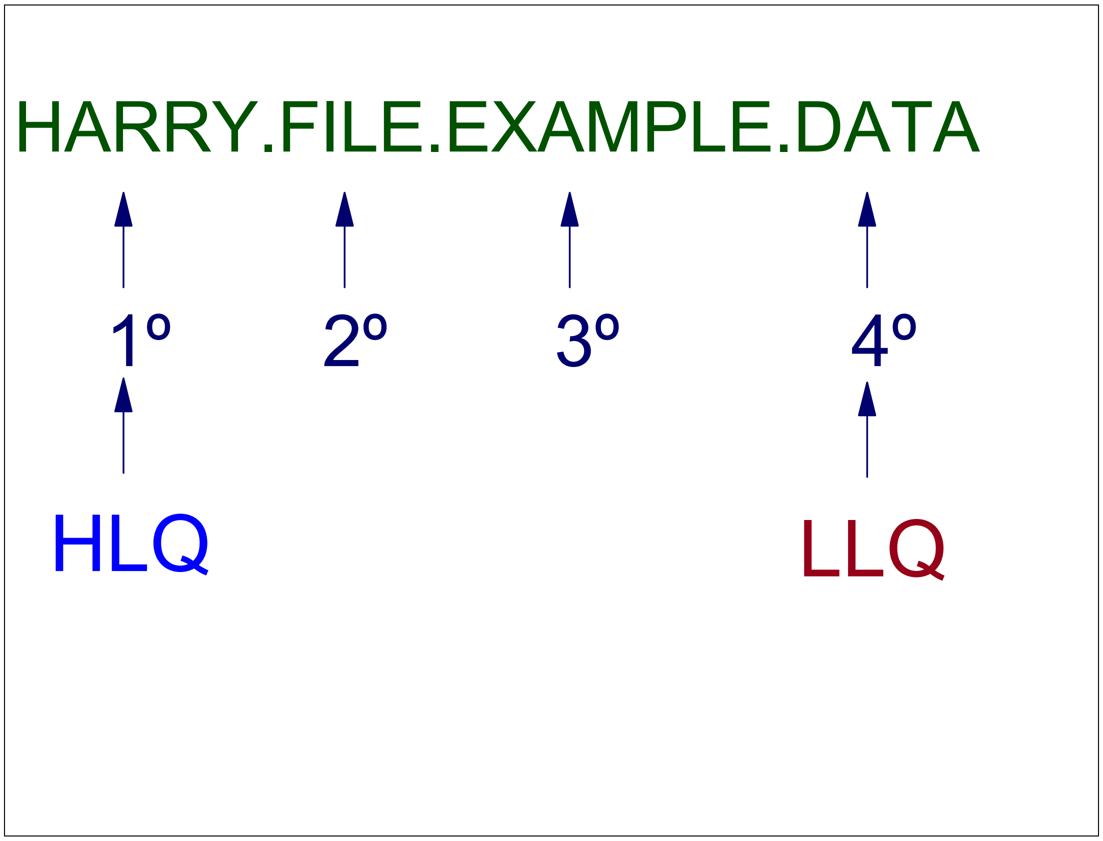


Figure 2-2 Data set name rules

Data set naming rules

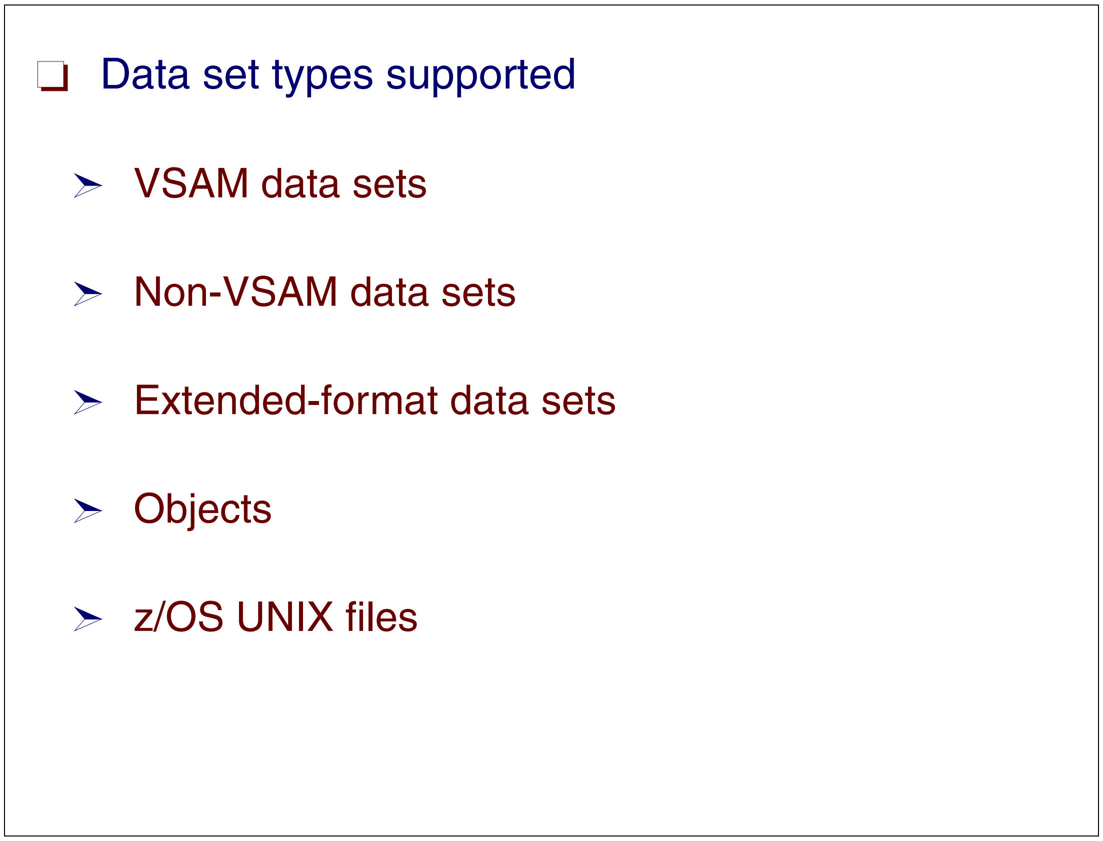
Whenever you allocate a new data set, you (or MVS) must give the data set a unique name. Usually, the data set name is given as the DSNAME keyword in JCL.

A data set name can be one name segment, or a series of joined name segments. Each name segment represents a level of qualification. For example, the data set name HARRY.FILE.EXAMPLE.DATA is composed of four name segments. The first name on the left is called the high-level qualifier (HLQ), the last name on the right is the lowest-level qualifier (LLQ).

Each name segment (qualifier) is 1 to 8 characters, the first of which must be alphabetic (A to Z) or national (# @ $). The remaining seven characters are either alphabetic, numeric (0 - 9), national, a hyphen (-). Name segments are separated by a period (.).

Note: Including all name segments and periods, the length of the data set name must not exceed 44 characters. Thus, a maximum of 22 name segments can make up a data set name.

2.3 DFSMSdfp data set types



- ❑ Data set types supported
  - ➢ VSAM data sets
  - ➢ Non-VSAM data sets
  - ➢ Extended-format data sets
  - ➢ Objects
  - ➢ z/OS UNIX files

Figure 2-3 DFSMSdfp data set types supported

DFSMSdfp data set types

The data organization that you choose depends on your applications and the operating environment. z/OS allows you to use temporary or permanent data sets, and to use several ways to organize files for data to be stored on magnetic media, as described here.

VSAM data sets

VSAM data sets are formatted differently than non-VSAM data sets. Except for linear data sets, VSAM data sets are collections of records, grouped into control intervals. The *control interval* is a fixed area of storage space in which VSAM stores records. The control intervals are grouped into contiguous areas of storage called *control areas*. To access VSAM data sets, use the VSAM access method. See also 2.4, “Types of VSAM data sets” on page 22.

Non-VSAM data sets

Non-VSAM data sets are collections of fixed-length or variable-length records, grouped into blocks, but not in control intervals. To access non-VSAM data sets, use BSAM, QSAM, or BPAM. See also 2.5, “Non-VSAM data sets” on page 23.

Extended-format data sets

You can create both sequential and VSAM data sets in extended format on system-managed DASD; this implies a 32-bye suffix at each physical record. See also “Extended-format data sets and objects” on page 25.

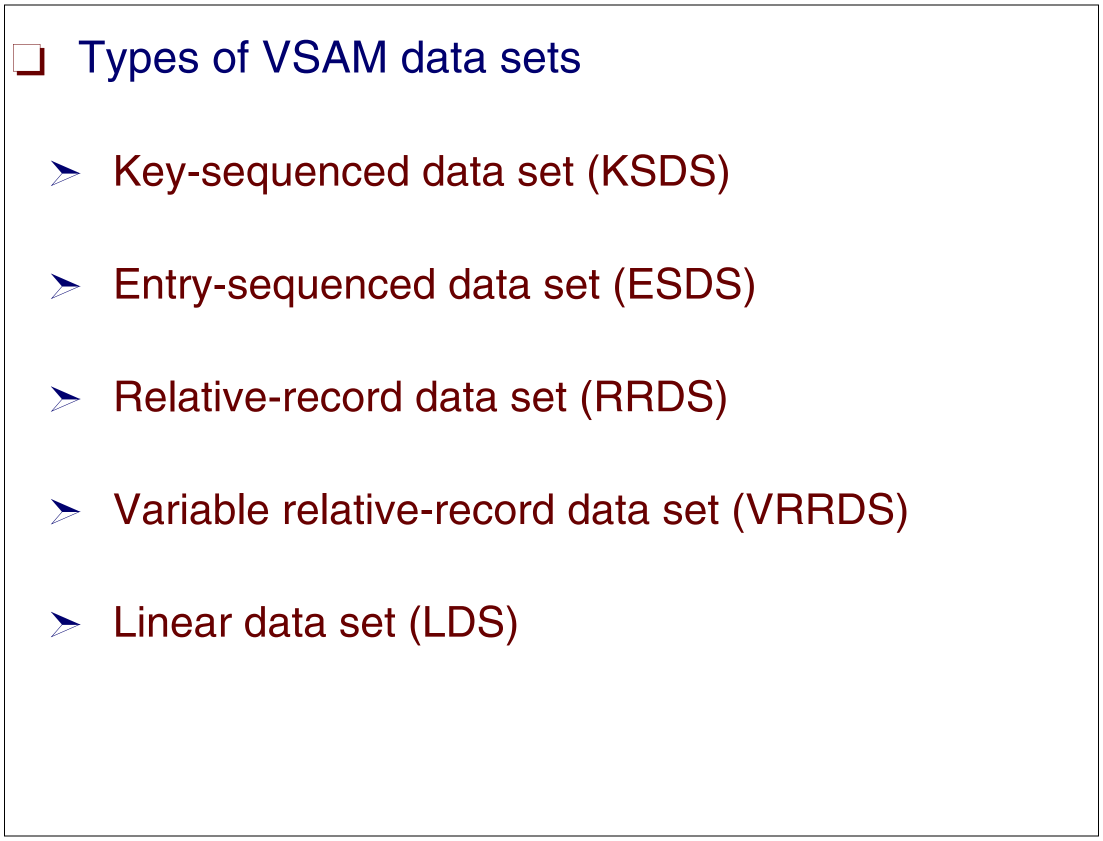
Objects

Objects are named streams of bytes that have no specific format or record orientation. Use the object access method (OAM) to store, access, and manage object data such as images.

z/OS UNIX files

z/OS UNIX System Services (z/OS UNIX) enables applications and even z/OS to access UNIX files. Also UNIX applications also can access z/OS data sets. You can use the hierarchical file system (HFS), z/OS Network File System (z/OS NFS), zSeries File System (zFS), and temporary file system (TFS) with z/OS UNIX. You can use the BSAM, QSAM, BPAM, and VSAM access methods to access data in UNIX files and directories. z/OS UNIX files are byte-oriented, similar to objects.

2.4 Types of VSAM data sets



- Types of VSAM data sets
  - Key-sequenced data set (KSDS)
  - Entry-sequenced data set (ESDS)
  - Relative-record data set (RRDS)
  - Variable relative-record data set (VRRDS)
  - Linear data set (LDS)

Figure 2-4 VSAM data set types

VSAM data sets

VSAM arranges records by an index key, by a relative byte address, or by a relative record number. VSAM data sets are cataloged for easy retrieval.

Key-sequenced data set (KSDS)

A KSDS VSAM data set contains records in order by a key field and can be accessed by the key or by a relative byte address. The key contains a unique value, such as an employee number or part number.

Entry-sequenced data set (ESDS)

An ESDS VSAM data set contains records in the order in which they were entered and can only be accessed by relative byte address. An ESDS is similar to a sequential data set.

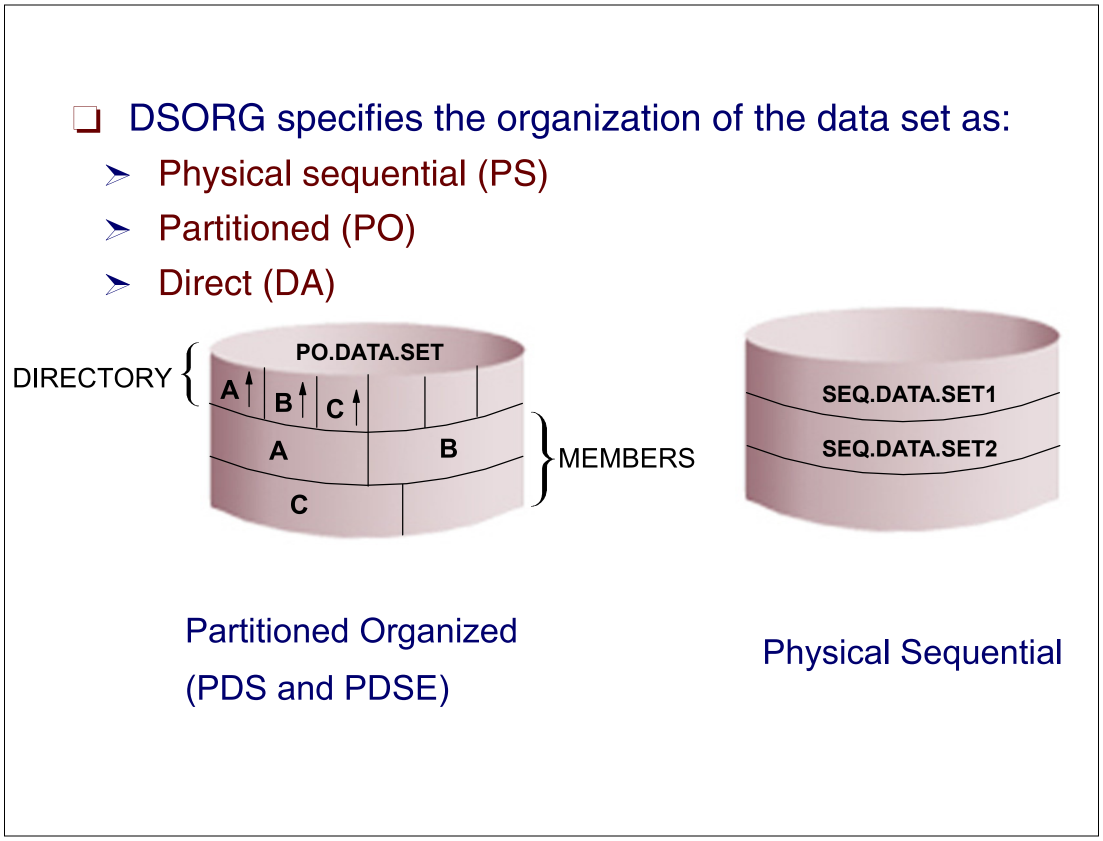
Relative-record data set (RRDS)

An RRDS VSAM data set contains records in order by relative-record number and can only be accessed by this number. Relative records can be fixed length or variable length. VRRDS is a type of RRDS where the logical records can be variable.

Linear data set (LDS)

An LDS VSAM data set contains data that can be accessed as byte-addressable strings in virtual storage. A linear data set does not have imbedded control information that other VSAM data sets hold.

2.5 Non-VSAM data sets



- DSORG specifies the organization of the data set as:
  - Physical sequential (PS)
  - Partitioned (PO)
  - Direct (DA)

Figure 2-5 Types of non-VSAM data sets

Data set organization (DSORG)

DSORG specifies the organization of the data set as physical sequential (PS), partitioned (PO), or direct (DA). If the data set is processed using absolute rather than relative addresses, you must mark it as unmovable by adding a U to the DSORG parameter (for example, by coding DSORG=PSU). You must specify the data set organization in the DCB macro. In addition:

- When creating a direct data set, the DSORG in the DCB macro must specify PS or PSU and the DD statement must specify DA or DAU.
- PS is for sequential and extended format DSNTYPE.
- PO is the data set organization for both PDSEs and PDSs. DSNTYPE is used to distinguish between PDSEs and PDSs.

Non-VSAM data sets

Non-VSAM data sets are collections of fixed-length or variable-length records grouped into physical blocks (a set of logical records). To access non-VSAM data sets, an application can use BSAM, QSAM, or BPAM. There are several types of non-VSAM data sets, as follows:

Physical sequential data set (PS)

Sequential data sets contain logical records that are stored in physical order. New records are appended to the end of the data set. You can specify a sequential data set in extended format or not.

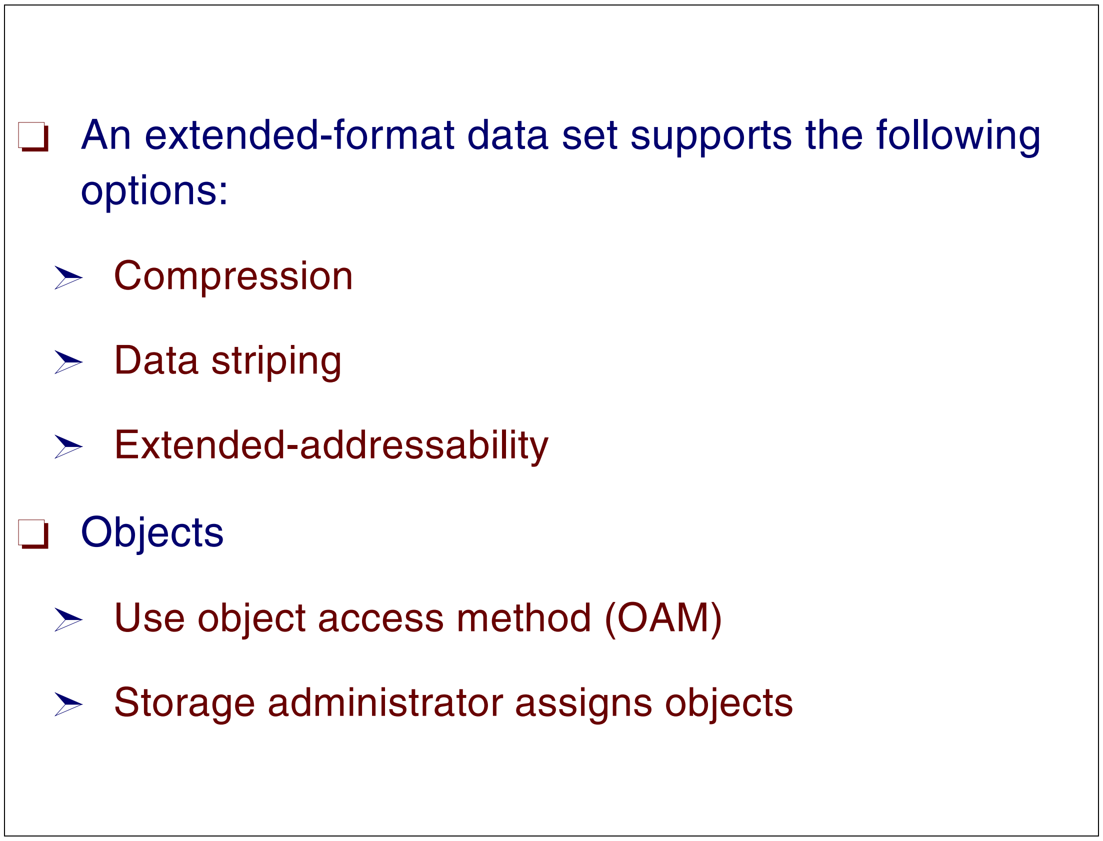
Partitioned data set (PDS)

Partitioned data sets contain a directory of sequentially organized members, each of which can contain a program or data. After opening the data set, you can retrieve any individual member without searching the entire data set.

Partitioned data set extended (PDSE)

Partitioned data sets extended contain an indexed, expandable directory of sequentially organized members, each of which can contain a program or data. You can use a PDSE instead of a PDS. The main advantage of using a PDSE over a PDS is that a PDSE automatically reclaims the space released by a previous member deletion, without the need for a reorganization.

2.6 Extended-format data sets and objects



- ❑ An extended-format data set supports the following options:
  - ➢ Compression
  - ➢ Data striping
  - ➢ Extended-addressability
- ❑ Objects
  - ➢ Use object access method (OAM)
  - ➢ Storage administrator assigns objects

Figure 2-6 Types of extended-format data sets

Extended-format data sets

While sequential data sets have a maximum of 16 extents on each volume, extended-format sequential data sets have a maximum of 123 extents on each volume. Each extended-format sequential data set can have a maximum of 59 volumes, so an extended-format sequential data set can have a maximum of 7257 extents (123 times 59).

An extended-format data set can occupy any number of tracks. On a volume that has more than 65,535 tracks, a sequential data set cannot occupy more than 65,535 tracks.

An extended-format, striped sequential data set can contain up to 4 GB blocks. The maximum size of each block is 32 760 bytes.

An extended-format data set supports the following additional functions:

- Compression, which reduces the space for storing data and improves I/O, caching, and buffering performance.
- Data striping, which in a sequential processing environment distributes data for one data set across multiple SMS-managed DASD volumes, improving I/O performance and reducing the batch window. For example, a data set with 6 stripes is distributed originally across 6 volumes.

  Large data sets with high sequential I/O activity are the best candidates for striped data sets. Data sets defined as extended-format sequential must be accessed using BSAM or QSAM, and *not* EXCP (means no access method is used) or BDAM.

- Extended-addressability, which enables you to create a VSAM data set that is larger than 4 GB.

System-managed DASD

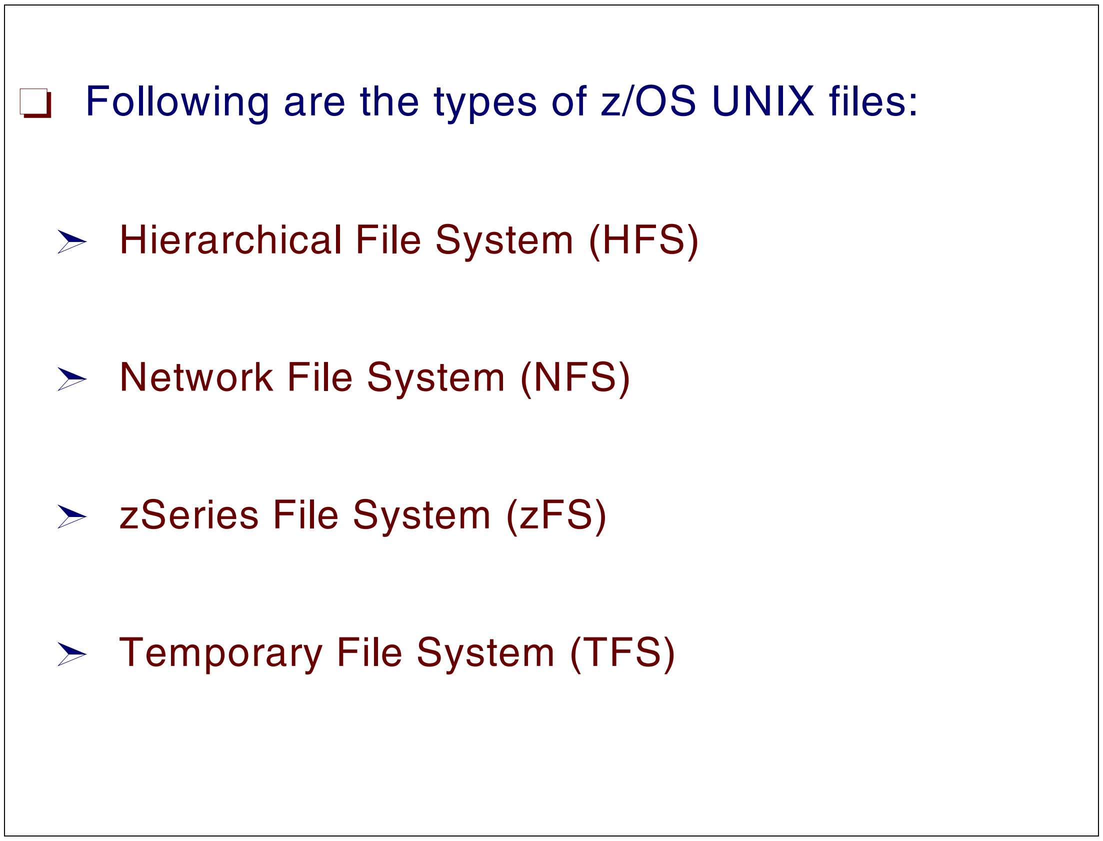
You can allocate both sequential and VSAM data sets in extended format on a system-managed DASD. Extended-format VSAM data sets also allow you to release partial unused space and to use system-managed buffering (SMB, a fast buffer pool management technique) for VSAM batch programs. You can select whether to use the primary or secondary space amount when extending VSAM data sets to multiple volumes.

Objects

Objects are named streams of bytes that have no specific format or record orientation. Use the object access method (OAM) to store, access, and manage object data. You can use any type of data in an object because OAM does not recognize the content, format, or structure of the data. For example, an object can be a scanned image of a document, an engineering drawing, or a digital video. OAM objects are stored either on DASD in a DB2® database, or on an optical drive, or on a tape storage volume.

The storage administrator assigns objects to object storage groups and object backup storage groups. The object storage groups direct the objects to specific DASD, optical, or tape devices, depending on their performance requirements. You can have one primary copy of an object, and up to two backup copies of an object.

2.7 z/OS UNIX files



- Following are the types of z/OS UNIX files:
  - Hierarchical File System (HFS)
  - Network File System (NFS)
  - zSeries File System (zFS)
  - Temporary File System (TFS)

Figure 2-7 z/OS UNIX files

z/OS UNIX

z/OS UNIX System Services (z/OS UNIX) enables z/OS to access UNIX files. UNIX applications also can access z/OS data sets. z/OS UNIX files are byte-oriented, similar to objects. We differentiate between the following types of z/OS UNIX files.

Hierarchical file system (HFS)

You can define on DASD an HFS data set on the z/OS system. Each HFS data set contains a hierarchical file system. Each hierarchical file system is structured like a tree with subtrees, and consists of directories and all their related files.

z/OS Network File System (z/OS NFS)

z/OS NFS is a distributed file system that enables users to access UNIX files and directories that are located on remote computers as though they were local. z/OS NFS is independent of machine types, operating systems, and network architectures.

zSeries File System (zFS)

A zFS is a UNIX file system that contains one or more file systems in a VSAM linear data set. zFS is application compatible with HFS and more performance efficient than HFS.

Temporary file system (TFS)

A TFS is stored in memory and delivers high-speed I/O. A systems programmer can use a TFS for storing temporary files.

2.8 Data set specifications for non-VSAM data sets

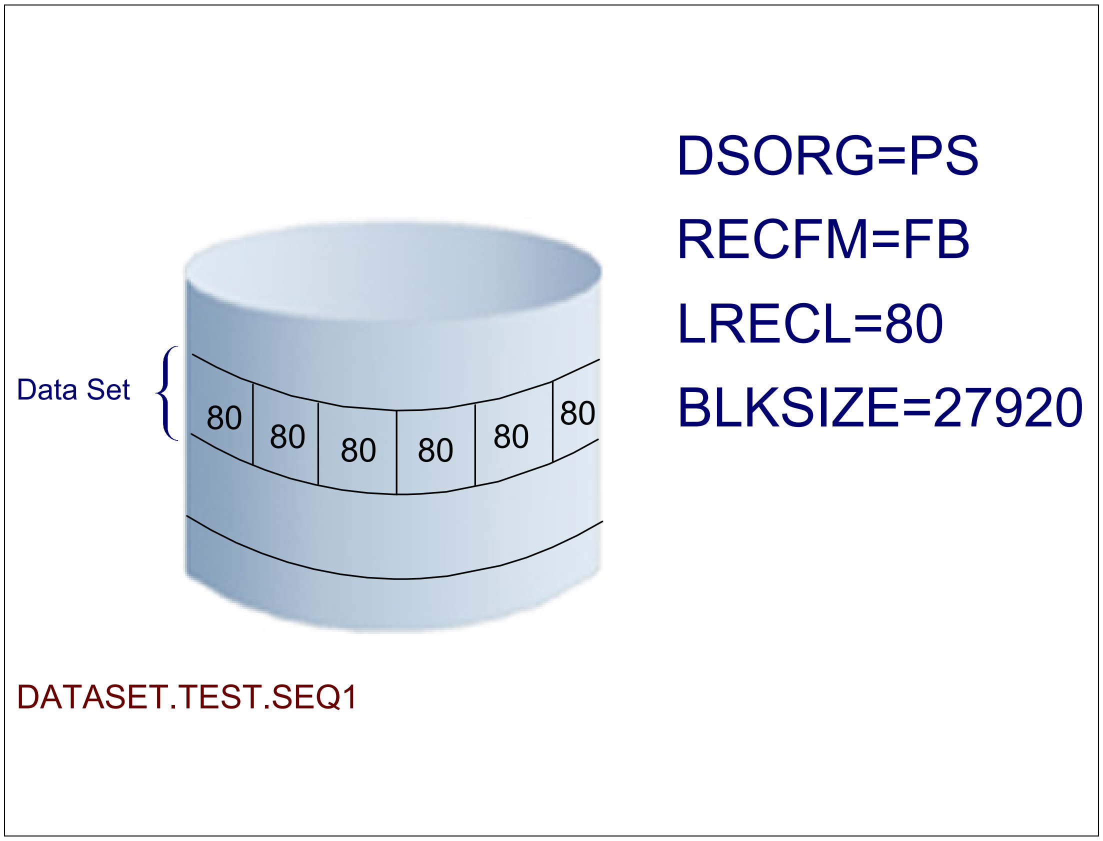


Figure 2-8 Data set specifications for non-VSAM data sets

Data set specifications for non-VSAM data set

A non-VSAM data set has several attributes that describe the data set. When you want to define (create) a new data set, you have to specify those values to tell the system which kind of data set you want to allocate.

See also *z/OS MVS JCL Reference,* SA22-7597 for information about the data set specifications discussed in this section.

Data set organization (DSORG)

DSORG specifies the organization of the data set as physical sequential (PS), partitioned (PO) for PDS or partitioned PDSE, or direct (DA).

Data set name type (DSNTYPE)

Use the DSNTYPE parameter to specify:

- For a partitioned organized (PO) data set, whether it is a:
  - PDS for a partitioned data set
  - LIBRARY for a partitioned data set extended (PDSE)
- Hierarchical file system if the DSNTYPE is HFS
- Large data sets (see “Volume table of contents (VTOC)” on page 36 for more details about large data sets)

Logical records and blocks

To an application, a *logical record* is a unit of information (for example, a customer, an account, a payroll employee, and so on). It is the smallest amount of data to be processed, and it is comprised of fields that contain information recognized by the processing application.

Logical records, when located in DASD or tape, are grouped into physical records named *blocks* (to save space in DASD because of the gaps). Each block of data on a DASD volume has a distinct location and a unique address (block number, track, and cylinder), thus making it possible to find any block without extensive sequential searching. Logical records can be stored and retrieved either directly or sequentially.

DASD volumes are used for storing data and executable programs (including the operating system itself), and for temporary working storage. One DASD volume can be used for many different data sets, and space on it can be reallocated and reused. The maximum length of a logical record (LRECL) is limited by the physical size of the media used.

Record format (RECFM)

RECFM specifies the characteristics of the logical records in the data set. They can have a:

- Fixed length (RECFM=F) - Every record is the same size.
- Variable length (RECFM=V) - The logical records can be of different sizes; every record has a preceding *record descriptor word* (RDW) to describe the length of such record.
- ASCII variable length (RECFM=D) - Used for ISO/ANSI tape data sets.
- Undefined length (RECFM=U) - Permits processing of records that do not conform to the F or V format.

F, V, or D-format logical records can be blocked (RECFM=FB, VB or DB), which means several logical records are in the same block.

Spanned records are specified as VS, VBS, DS, or DBS. A *spanned record* is a logical record that spans two or more blocks. Spanned records can be necessary if the logical record size is larger than the maximum allowed block size.

You can also specify the records as fixed-length standard by using FS or FBS, meaning that there is not an internal short block.

Logical record length (LRECL)

LRECL specifies the length, in bytes, of each record in the data set. If the records are of variable length or undefined length, LRECL specifies the maximum record length. For input, the field has no effect for undefined-length (format-U) records.

Block size (BLKSIZE)

BLKSIZE specifies the maximum length, in bytes, of the physical record (block). If the logical records are format-F, the block size must be an integral multiple of the record length. If the records are format-V, you must specify the *maximum block size*. If format-V records are *unblocked*, the block size must be 4 bytes (to specify the block length) greater than the record length (LRECL). For data sets in DASD the maximum block size is 32760. For data sets in tapes the maximum block size is much larger.

In an extended-format data set, the system adds a 32-byte suffix to each block, which is transparent to the application program.

System-determined block size

The system can derive the best block size to optimize DASD space, tape, and spooled data sets (the ones in printing format but stored temporarily in DASD).

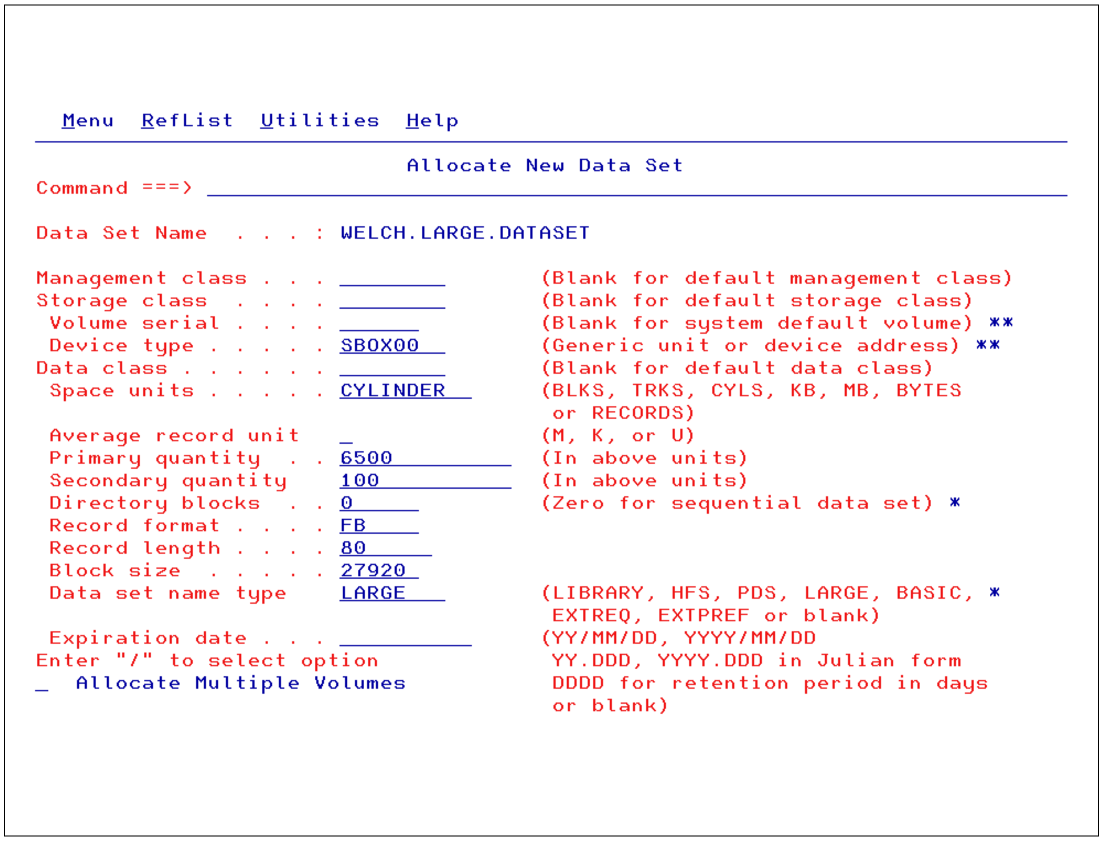
Space values

For DASD data sets, you can specify the amount of space required in: logical records, blocks, records, tracks, or cylinders. You can specify a primary and a secondary space allocation. When you define a new data set, only the primary allocation value is used to reserve space for the data set on DASD. Later, when the primary allocation of space is filled, space is allocated in secondary storage amounts (if specified). The extents can be allocated on other volumes if the data set was defined as multivolume.

For example, if you allocate a new data set and specify SPACE=(TRK,(2,4)), this initially allocates two tracks for the data set. As each record is written to the data set and these two tracks are used up, the system automatically obtains four more tracks. When these four tracks are used, another four tracks are obtained. The same sequence is followed until the extent limit for the type of data set is reached.

The procedure for allocating space on magnetic tape devices is different from allocating space on DASD. Because data sets on magnetic tape devices must be organized sequentially, each one is located contiguously. All data sets that are stored on a given magnetic tape volume must be recorded in the same density. See *z/OS DFSMS Using Magnetic Tapes,* SC26-7412 for information about magnetic tape volume labels and tape processing.

2.9 Large format data sets



```
  Menu  RefList  Utilities  Help
 ------------------------------------------------------------------------------
                           Allocate New Data Set
Command ===> _________________________________________________________________

Data Set Name  . . . : WELCH.LARGE.DATASET

Management class . . . ________        (Blank for default management class)
Storage class  . . . . ________        (Blank for default storage class)
 Volume serial . . . . ______          (Blank for system default volume) **
 Device type . . . . . SBOX00          (Generic unit or device address) **
Data class . . . . . . ________        (Blank for default data class)
 Space units . . . . . CYLINDER        (BLKS, TRKS, CYLS, KB, MB, BYTES
                                        or RECORDS)
 Average record unit   _               (M, K, or U)
 Primary quantity  . . 6500            (In above units)
 Secondary quantity    100             (In above units)
 Directory blocks  . . 0               (Zero for sequential data set) *
 Record format . . . . FB
 Record length . . . . 80
 Block size  . . . . . 27920
 Data set name type    LARGE           (LIBRARY, HFS, PDS, LARGE, BASIC, *
                                        EXTREQ, EXTPREF or blank)
 Expiration date . . . __________      (YY/MM/DD, YYYY/MM/DD
Enter "/" to select option              YY.DDD, YYYY.DDD in Julian form
_  Allocate Multiple Volumes            DDDD for retention period in days
                                        or blank)
```

Figure 2-9 Allocating a data set with ISPF option 3.2

Large format data sets

Defining large format data sets was introduced with z/OS V1R7. Large format data sets are physical sequential data sets, with generally the same characteristics as other non-extended format sequential data sets, but with the capability to grow beyond the basic format size limit of 65535 tracks on each volume. (This is about 3,500,000,000 bytes, depending on the block size.) Large format data sets reduce the need to use multiple volumes for single data sets, especially very large ones like spool data sets, dumps, logs, and traces. Unlike extended-format data sets, which also support greater than 65535 tracks per volume, large format data sets are compatible with EXCP and do not need to be SMS-managed.

Data sets defined as large format must be accessed using QSAM, BSAM, or EXCP.

Large format data sets have a maximum of 16 extents on each volume. Each large format data set can have a maximum of 59 volumes. Therefore, a large format data set can have a maximum of 944 extents (16 times 59).

A large format data set can occupy any number of tracks, without the limit of 65535 tracks per volume. The minimum size limit for a large format data set is the same as for other sequential data sets that contain data: one track, which is about 56 000 bytes. Primary and secondary space can both exceed 65 535 tracks per volume.

Large format data sets can be on SMS-managed DASD or non-SMS-managed DASD.

Restriction: The following types of data sets cannot be allocated as large format data sets:

- PDS, PDSE, and direct data sets
- Virtual I/O data sets, password data sets, and system dump data sets

Allocating data sets

To process an already existing data set, first allocate it (establish a logical link with it and your program), then access the data using macros in Assembler or HLL statements to activate the access method that you have chosen. The allocation of a data set means either or both of two things:

- To set aside (create) space for a new data set on a disk or tape
- To establish a logical link between a job step (your program) and any data set using JCL

Figure 2-9 on page 31 shows the creation of a data set using the ISPF panel 3.2. Other ways to create a data set are by using any of the following methods:

- Access method services

  You can define VSAM data sets and establish catalogs by using a multifunction services program called *access method services*.

- TSO ALLOCATE command

  You can issue the **ALLOCATE** command through TSO/E to define VSAM and non-VSAM data sets.

- Using JCL

  Any data set can be defined directly through JCL by specifying DSNTYPE=LARGE on the DD statement.

2.10 Locating an existing data set

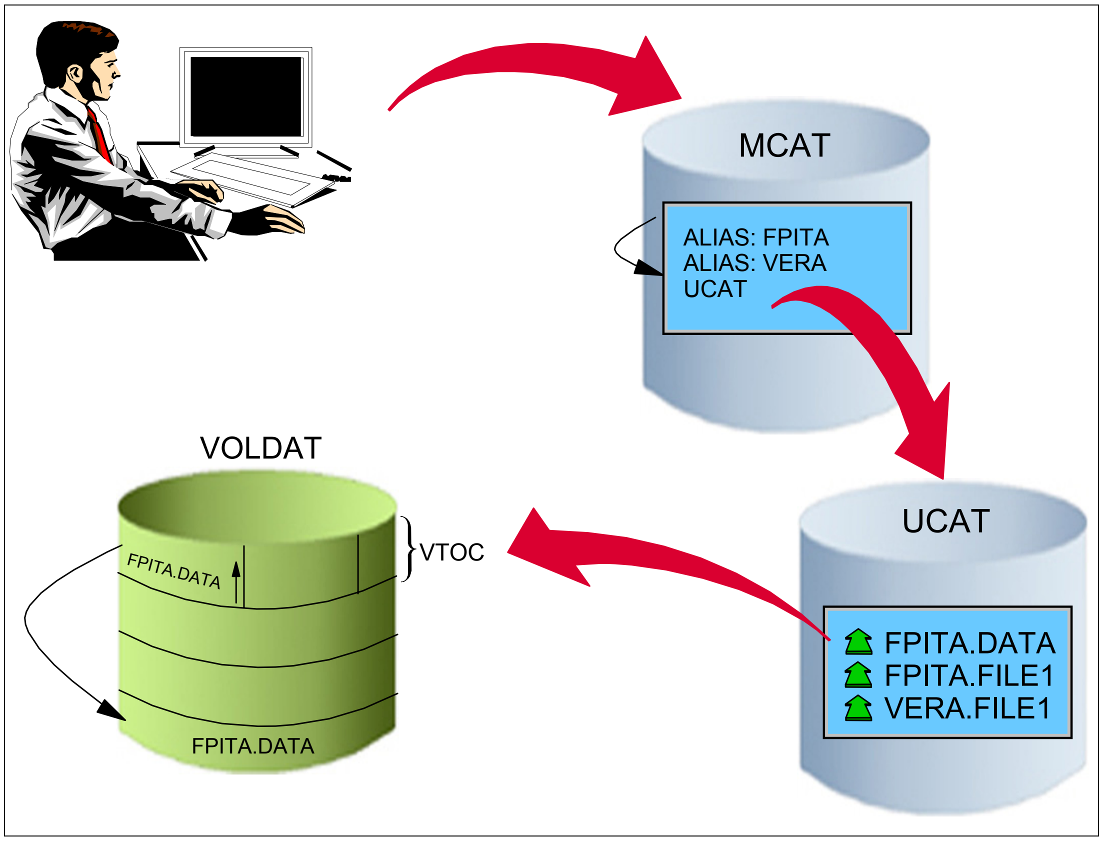


Figure 2-10 Locating an existing data set

Locating a data set

A catalog consists of two separate kinds of data sets: a basic catalog structure (BCS); and a VSAM volume data set (VVDS). The BCS can be considered the catalog, whereas the VVDS can be considered an extension of the volume table of contents (VTOC). Following are the terms used to locate a data set that has been cataloged:

VTOC	The volume table of contents is a sequential data set located in each DASD volume that describes the data set contents of this volume. The VTOC is used to find empty space for new allocations and to locate non-VSAM data sets. For all VSAM data sets, and for SMS-managed non-VSAM data sets, the VTOC is used to obtain information not kept in the VVDS. See "VTOC index structure" on page 39.
User catalog	A catalog is a data set used to locate in which DASD volume the requested data set is stored; user application data sets are cataloged in this type of catalog.
Master catalog	This has the same structure as a user catalog, but points to system (z/OS) data sets. It also contains information about the user catalog location and any alias pointer.
Alias	A special entry in the master catalog pointing to a user catalog that coincides with the HLQ of a data set name. The alias is used to find in which user catalog the data set location information exists. It means that the data set with this HLQ is cataloged in that user catalog.

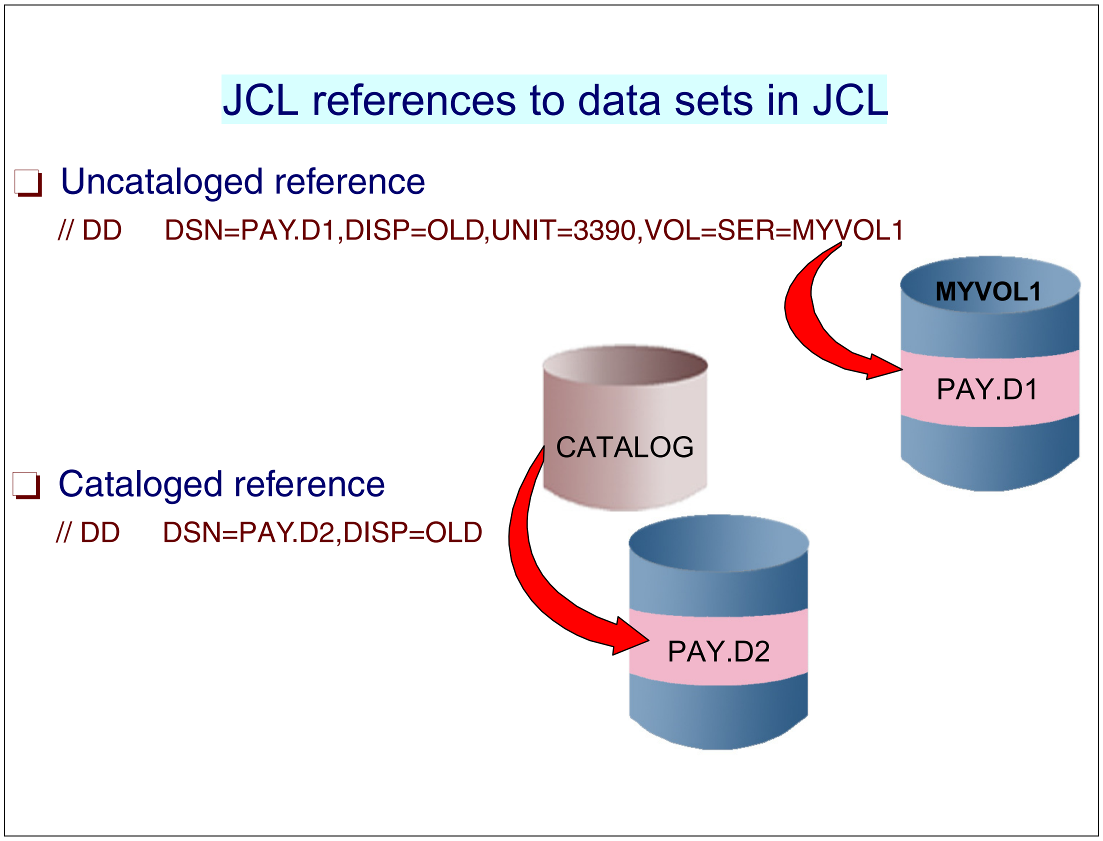
The sequence for locating an existing data set

The MVS system provides a service called LOCATE to read entries in a catalog. When z/OS tries to locate an existing data set, the following sequence takes place.

- When the master catalog is examined:
  - If it has the searched data set name, the volume information is picked up and the volume VTOC is used to locate the data set in the specified volume.
  - If the HLQ is a defined alias in the master catalog, the user catalog is searched. If the data set name is found, processing proceeds as in a master catalog find.
- Finally, the requesting program can access the data set. As you can imagine, it is impossible to keep track of the location of millions of data sets without the catalog concept.

For detailed information about catalogs refer to Chapter 6, “Catalogs” on page 305.

2.11 Uncataloged and cataloged data sets


JCL references to data sets in JCL

❑ Uncataloged reference

```
// DD     DSN=PAY.D1,DISP=OLD,UNIT=3390,VOL=SER=MYVOL1
```

❑ Cataloged reference

```
// DD     DSN=PAY.D2,DISP=OLD
```

Figure 2-11 Cataloged and uncataloged data sets

Cataloged data sets

When an existing data set is cataloged, z/OS obtains unit and volume information from the catalog using the LOCATE macro service. However, if the DD statement for a catalog data set contains VOLUME=SER=serial-number, the system does not look in the catalog; in this case, you must code the UNIT parameter and volume information.

Uncataloged data sets

When your existing data set is not cataloged, you *must* know in advance its volume location and specify it in your JCL. This can be done through the UNIT and VOL=SER, as shown in Figure 2-11.

See *z/OS MVS JCL Reference,* SA22-7597 for information about UNIT and VOL parameters.

Note: We strongly recommend that you do not have uncataloged data sets in your installation because uncataloged data sets can cause problems with duplicate data and possible incorrect data set processing.

2.12 Volume table of contents (VTOC)

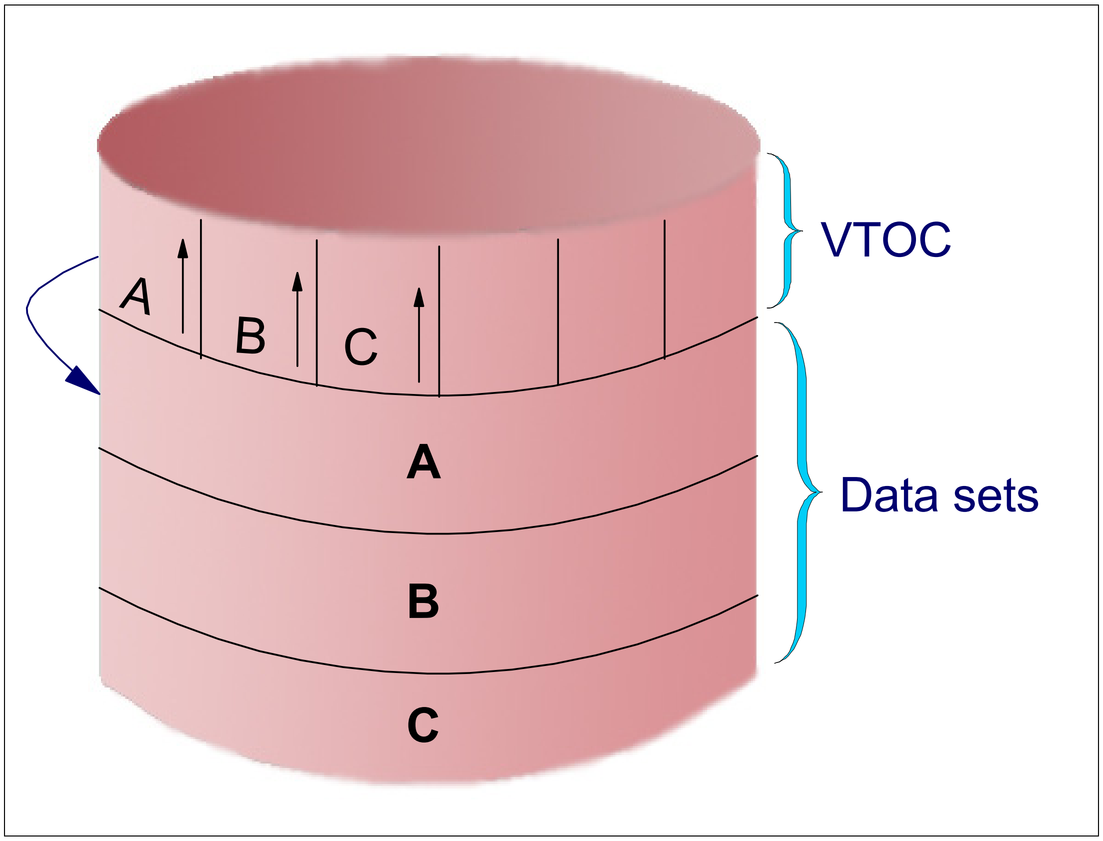


Figure 2-12 Volume table of contents (VTOC)

Volume table of contents (VTOC)

The VTOC lists the data sets that reside on its volume, along with information about the location and size of each data set, and other data set attributes. It is created when the volume is initialized through the ICKDSF utility program.

The VTOC locates data sets on that volume. The VTOC is composed of 140-byte data set control blocks (DSCBs), of which there are six types shown in Table 2-1 on page 38, that correspond either to a data set currently residing on the volume, or to contiguous, unassigned tracks on the volume. A set of assembler macros is used to allow a program or z/OS to access VTOC information.

IEHLIST utility

The IEHLIST utility can be used to list, partially or completely, entries in a specified volume table of contents (VTOC), whether indexed or non-indexed. The program lists the contents of selected data set control blocks (DSCBs) in edited or unedited form.

2.13 VTOC and DSCBs

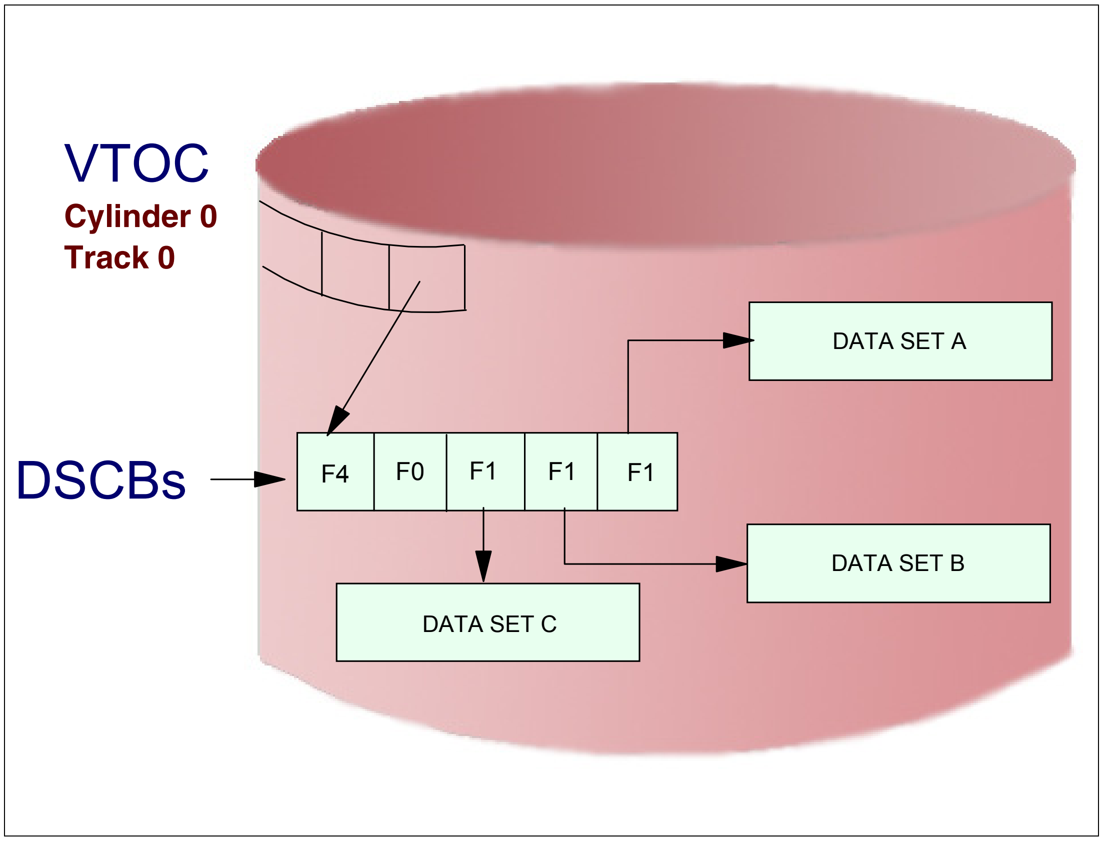


Figure 2-13 Data set control block (DSCB)

Data set control block (DSCB)

The VTOC is composed of 140-byte data set control blocks (DSCBs) that point to data sets currently residing on the volume, or to contiguous, unassigned (free) tracks on the volume (depending on the DSCB type).

DSCBs also describe the VTOC itself. CVAF routines automatically construct a DSCB when space is requested for a data set on the volume. Each data set on a DASD volume has one or more DSCBs (depending on its number of extents) describing space allocation and other control information such as operating system data, device-dependent information, and data set characteristics. There are seven kinds of DSCBs, each with a different purpose and a different format number.

The first record in every VTOC is the VTOC DSCB (format-4). The record describes the device, the volume the data set resides on, the volume attributes, and the size and contents of the VTOC data set itself. The next DSCB in the VTOC data set is a free-space DSCB (format-5) that describes the unassigned (free) space in the full volume.

The function of some DSCBs depends on whether an optional Index VTOC is allocated in the volume. Index VTOC is a sort of B-tree, to make the search in VTOC faster.

Table 2-1 on page 38 describes the different types of DSCBs, taking into consideration whether the Index VTOC is in place or not.

In z/OS 1.7 there is a new AS (DEVMAN) containing trace information about CVAF events.

Table 2-1 DSCBs that can be found in the VTOC

Type	Name	Function	How many
0	Free VTOC DSCB	Describes unused DSCB records in the VTOC (contains 140 bytes of binary zeros). To delete a DSCB from the VTOC, a format-0 DSCB is written over it.	One for every unused 140-byte record in the VTOC. The DS4DSREC field of the format-4 DSCB is a count of the number of format-0 DSCBs in the VTOC. This field is not maintained for an indexed VTOC.
1	Identifier	Describes the first three extents of a data set or VSAM data space.	One for every data set or data space on the volume, except the VTOC.
2	Index	Describes the indexes of an ISAM data set. This data set organization is old, and is not supported anymore.	One for each ISAM data set (for a multivolume ISAM data set, a format-2 DSCB exists only on the first volume).
3	Extension	Describes extents after the third extent of a non-VSAM data set or a VSAM data space.	One for each data set on the volume that has more than three extents. There can be as many as 10 for a PDSE, HFS, extended format data set, or a VSAM data set component cataloged in an integrated catalog facility catalog. PDSEs, HFS, and extended format data sets can have up to 123 extents per volume. All other data sets are restricted to 16 extents per volume. A VSAM component can have 7257 extents in up to 59 volumes (123 each).
4	VTOC	Describes the extent and contents of the VTOC, and provides volume and device characteristics. This DSCB contains a flag indicating whether the volume is SMS-managed.	One on each volume.
5	Free space	On a non-indexed VTOC, describes the space on a volume that has not been allocated to a data set (available space). For an indexed VTOC, a single empty format-5 DSCB resides in the VTOC; free space is described in the index and DS4IVTOC is normally on.	One for every 26 non-contagious extents of available space on the volume for a non- indexed VTOC; for an indexed VTOC, there is only one.
7	Free space for certain device	Only one field in the format-7 DSCB is an intended interface. This field indicates whether the DSCB is a format-7 DSCB. You can reference that field as DS1FMTID or DS5FMTID. A character 7 indicates that the DSCB is a format-7 DSCB, and your program should not modify it.	This DSCB is not used frequently.

2.14 VTOC index structure

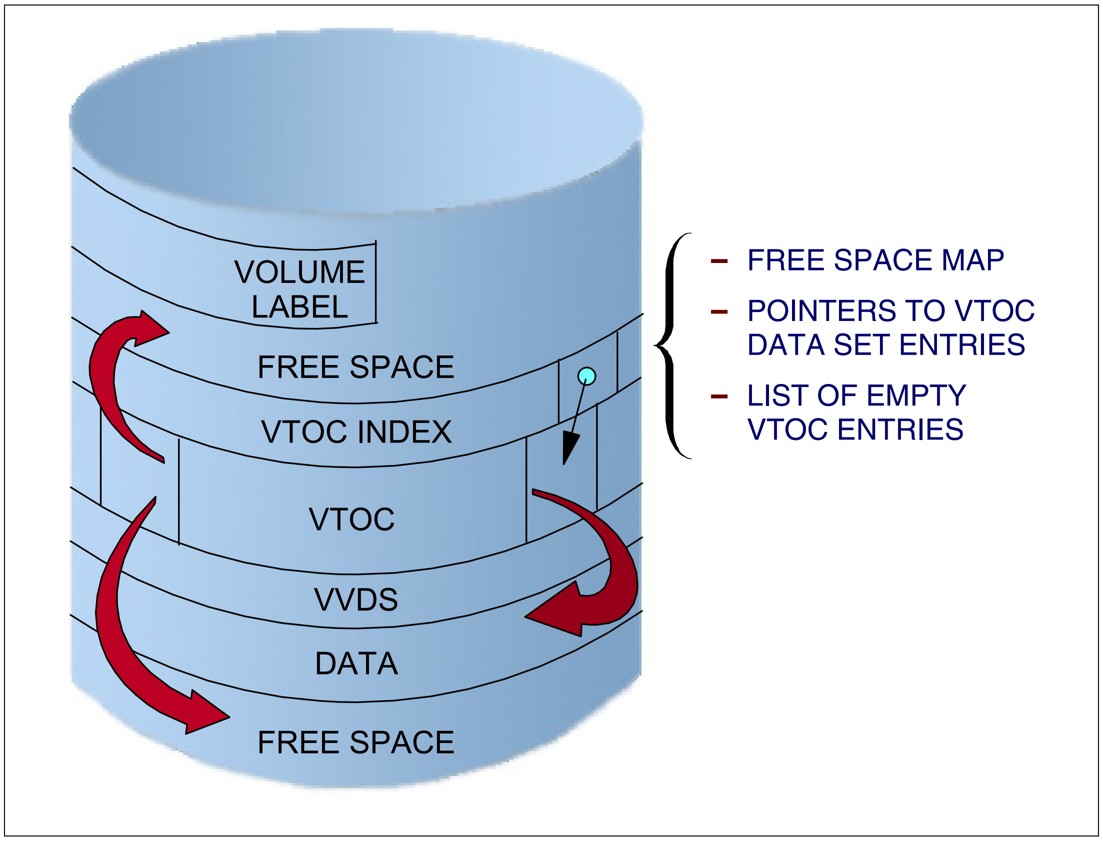


Figure 2-14 VTOC index structure

VTOC index

The VTOC index enhances the performance of VTOC access. The VTOC index is a physical-sequential data set on the same volume as the related VTOC, created by the ICKDSF utility program. It consists of an index of data set names in format-1 DSCBs contained in the VTOC and volume free space information.

Important: An SMS-managed volume *requires* an indexed VTOC; otherwise, the VTOC index is highly recommended. For additional information about SMS-managed volumes, see *z/OS DFSMS Implementing System-Managed Storage,* SC26-7407.

If the system detects a logical or physical error in a VTOC index, the system disables further access to the index from all systems that might be sharing the volume. Then, the VTOC remains usable but with possibly degraded performance.

If a VTOC index becomes disabled, you can rebuild the index without taking the volume offline to any system. All systems can continue to use that volume without interruption to other applications, except for a brief pause during the index rebuild. After the system rebuilds the VTOC index, it automatically re-enables the index on each system that has access to it.

Next, we see more details about the internal implementation of the Index VTOC.

Creating the VTOC and VTOC index

To initialize a volume (prepare for I/O activity), use the Device Support Facilities (ICKDSF) utility to initially build the VTOC. You can create a VTOC index at that time by using the ICKDSF INIT command and specifying the INDEX keyword.

You can use ICKDSF to convert a non-indexed VTOC to an indexed VTOC by using the BUILDIX command and specifying the IXVTOC keyword. The reverse operation can be performed by using the BUILDIX command and specifying the OSVTOC keyword. For details, see *Device Support Facilities User's Guide and Reference Release 17,* GC35-0033, and *z/OS DFSMSdfp Advanced Services,* SC26-7400, for more information about that topic.

VTOC index format-1 DSCB

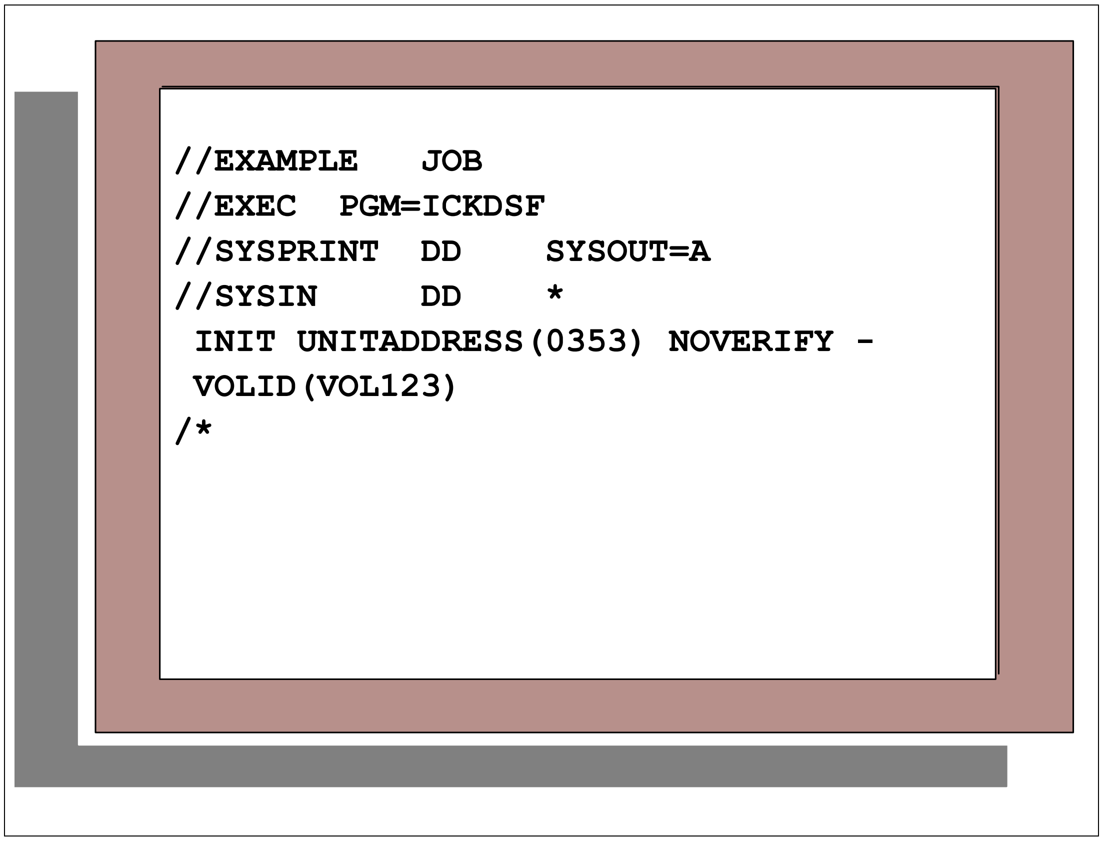
A format-1 DSCB in the VTOC contains the name and extent information of the VTOC index. The name of the index must be 'SYS1.VTOCIX.*xxxxxxxx*', where *xxxxxxxx* conforms to standard data set naming conventions and is usually the serial number of the volume containing the VTOC and its index. The name must be unique within the system to avoid ENQ contention.

VTOC index record (VIR)

Device Support Facilities (ICKDSF) initializes a VTOC index into 2048-byte physical blocks named VTOC index records (VIRs). VIRs are used in several ways. A VTOC index contains the following kinds of VIRs:

- VTOC index entry record (VIER) identifies the location of format-1 DSCBs and the format-4 DSCB.
- VTOC pack space map (VPSM) identifies the free and allocated space on a volume.
- VTOC index map (VIXM) identifies the VIRs that have been allocated in the VTOC index.
- VTOC map of DSCBs (VMDS) identifies the DSCBs that have been allocated in the VTOC.

2.15 Initializing a volume using ICKDSF



```
//EXAMPLE   JOB
//EXEC  PGM=ICKDSF
//SYSPRINT  DD    SYSOUT=A
//SYSIN     DD    *
 INIT UNITADDRESS(0353) NOVERIFY -
 VOLID(VOL123)
/*
```

Figure 2-15 Initializing a volume

Device Support Facilities (ICKDSF)

ICKDSF is a program you can use to perform functions needed for the initialization, installation, use, and maintenance of DASD volumes. You can also use it to perform service functions, error detection, and media maintenance. However, due to the virtualization of the volumes there is no need for executing media maintenance.

Initializing a DASD volume

After you have completed the installation of a device, you must initialize and format the volume so that it can be used by MVS.

You use the INIT command to initialize volumes. The INIT command writes a volume label (on cylinder 0, track 0) and a VTOC on the device for use by MVS. It reserves and formats tracks for the VTOC at the location specified by the user and for the number of tracks specified. If no location is specified, tracks are reserved at the default location.

If the volume is SMS-managed, the *STORAGEGROUP* option must be declared, in order to keep such information (SMS managed) in a format-4 DSCB.

Initializing a volume for the first time in offline mode

In the example in Figure 2-15, a volume is initialized at the minimal level because neither the CHECK nor VALIDATE parameter is specified (as recommended). Because the volume is being initialized for the first time, it must be mounted offline (to avoid MVS data set allocations), and the volume serial number must be specified. Because the VTOC parameter is not specified, the default VTOC size is the number of tracks in a cylinder minus one. For a 3390, the default location is cylinder 0, track 1 for 14 tracks.

Initializing a volume to be managed in a DFSMS environment

In the following example, a volume that is to be system-managed is initialized. The volume is initialized in offline mode at the minimal level. The VTOC is placed at cylinder 2, track 1 and occupies ten tracks. The VTOC is followed by the VTOC index. The STORAGEGROUP parameter indicates the volume is to be managed in a DFSMS environment.

```
   INIT UNIT(0353) NOVERIFY STORAGEGROUP -
           OWNERID(PAYROLL) VTOC(2,1,10) INDEX(2,11,5)
```

The following example performs an online minimal initialization, and as a result of the command, an index to the VTOC is created.

```
//         JOB
//         EXEC  PGM=ICKDSF
//XYZ987   DD    UNIT=3390,DISP=OLD,VOL=SER=PAY456
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
 INIT DDNAME(XYZ987) NOVERIFY INDEX(X'A',X'B',X'2')
/*
```

ICKDSF stand-alone version

You can run the stand-alone version of ICKDSF under any IBM System z™ processor. To run the stand-alone version of ICKDSF, you IPL ICKDSF with a stand-alone IPL tape that you create under z/OS. This function allows you to initialize volumes without the need for running an operating system such as z/OS.

Creating an ICKDSF stand-alone IPL tape using z/OS

For z/OS, the stand-alone code is in SYS1.SAMPLIB as ICKSADSF. You can load the ICKDSF program from a file on tape. The following example can be used to copy the stand-alone program to an unlabeled tape.

```
//JOBNAME  JOB  JOB CARD PARAMETERS
//STEPNAME EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSIN    DD DUMMY,DCB=BLKSIZE=80
//SYSUT1   DD DSNAME=SYS1.SAMPLIB(ICKSADSF),UNIT=SYSDA,
//            DISP=SHR,VOLUME=SER=XXXXXX
//SYSUT2   DD DSNAME=ICKDSF,UNIT=3480,LABEL=(,NL),
//            DISP=(,KEEP),VOLUME=SER=YYYYYY,
//            DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
```

For details on how to IPL the stand-alone version and to see examples of the commands, refer to *Device Support Facilities User's Guide and Reference Release 17,* GC35-0033.

3

Storage management hardware

The use of DFSMS requires storage management hardware that includes both Direct Access Storage Devices (DASD) and tape devices. In this chapter we provide an overview of both storage device categories, as well as a brief introduction to RAID technology.

For many years, DASD devices have been the most used storage devices on IBM eServer™ zSeries systems and their predecessors, delivering the fast random access to data and high availability that customers have come to expect.

We cover the following types of DASD:

- Traditional DASD (such as 3380 and 3390)
- Enterprise Storage Server (ESS)
- DS6000™ and DS8000™

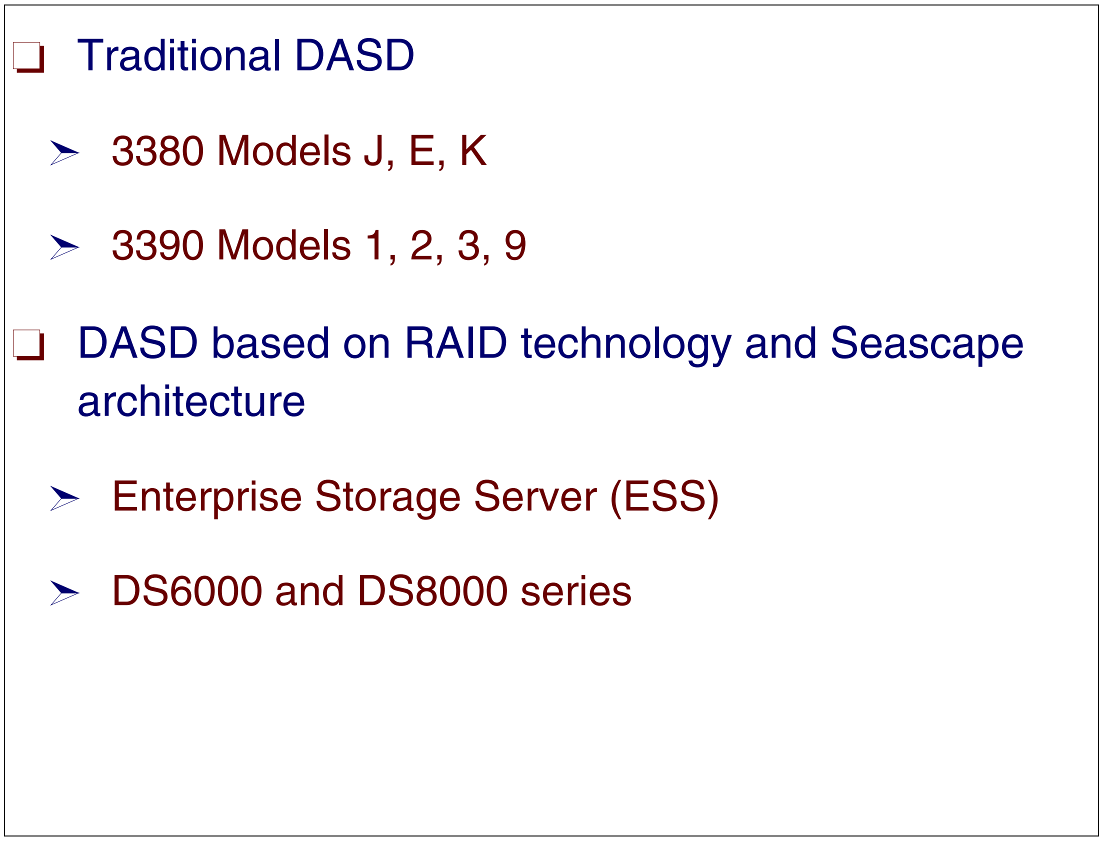
The era of tapes began before DASD was introduced. During that time, tapes were used as the primary application storage medium. Today customers use tapes for such purposes as backup, archiving, or data transfer between companies.

The following types of tape devices are described:

- Traditional tapes like 3480 and 3490
- IBM Magstar® 3590 and 3592
- Automated tape library (ATL) 3494
- Virtual tape server (VTS).

We also briefly explain the storage area network (SAN) concept.

3.1 Overview of DASD types



- Traditional DASD
  - 3380 Models J, E, K
  - 3390 Models 1, 2, 3, 9
- DASD based on RAID technology and Seascape architecture
  - Enterprise Storage Server (ESS)
  - DS6000 and DS8000 series

Figure 3-1 Overview of DASD types

Traditional DASD

In the era of traditional DASD, the hardware consisted of controllers like 3880 and 3990, which contained the necessary intelligent functions to operate a storage subsystem. The controllers were connected to S/390 systems via parallel or ESCON® channels. Behind a controller you had several model groups of the 3390 that contained the disk drives. Based on the models, these disk drives had different capacities per device. Within each model group, the different models provide either four, eight, or twelve devices. All A-units come with four controllers, providing a total of four paths to the 3990 Storage Control. At that time, you were not able to change the characteristics of a given DASD device.

DASD based on RAID technology

With the introduction of the RAMAC Array in 1994, IBM first introduced storage subsystems for S/390 systems based on RAID technology. We discuss the various RAID implementations in “Redundant array of independent disks (RAID)” on page 49.

The more modern IBM DASD products, such as Enterprise Storage Server (ESS), DS6000, DS8000, and DASD from other vendors, emulate IBM 3380 and 3390 volumes in geometry, capacity of tracks, and number of tracks per cylinder. This emulation makes all the other entities think they are dealing with real 3380s or 3390s. Among these entities, we have data processing people not working directly with storage, JCL, MVS commands, open routines, access methods, IOS, and channels. One advantage of this emulation is that it allows DASD manufacturers to implement changes in the real disks, including the geometry of tracks and cylinders, without affecting the way those components interface with DASD. From an operating system point of view, device types always will be 3390s, sometimes with much higher numbers of cylinders, but 3390s nonetheless.

ESS technology

The IBM TotalStorage Enterprise Storage Server (ESS) is IBM's disk storage server, developed using IBM Seascape® architecture. The ESS provides functionality to the family of e-business servers, and also to non-IBM (that is, Intel®-based and UNIX-based) families of servers. Across all of these environments, the ESS features unique capabilities that allow it to meet the most demanding requirements of performance, capacity, and data availability that the computing business may require. See "Enterprise Storage Server (ESS)" on page 54 for more information about this topic.

Seascape architecture

The Seascape architecture is the key to the development of IBM's storage products. Seascape allows IBM to take the best of the technologies developed by the many IBM laboratories and integrate them, producing flexible and upgradeable storage solutions. This Seascape architecture design has allowed the IBM TotalStorage Enterprise Storage Server to evolve from the initial E models to the succeeding F models, and to the later 800 models, each featuring new, more powerful hardware and functional enhancements, and always integrated under the same successful architecture with which the ESS was originally conceived. Refer to "Seascape architecture" on page 51 for more information.

Note: In this publication, we use the terms *disk* or *head disk assembly* (HDA) for the real devices, and the terms *DASD volumes* or *DASD devices* for the logical 3380/3390s.

3.2 Traditional DASD capacity

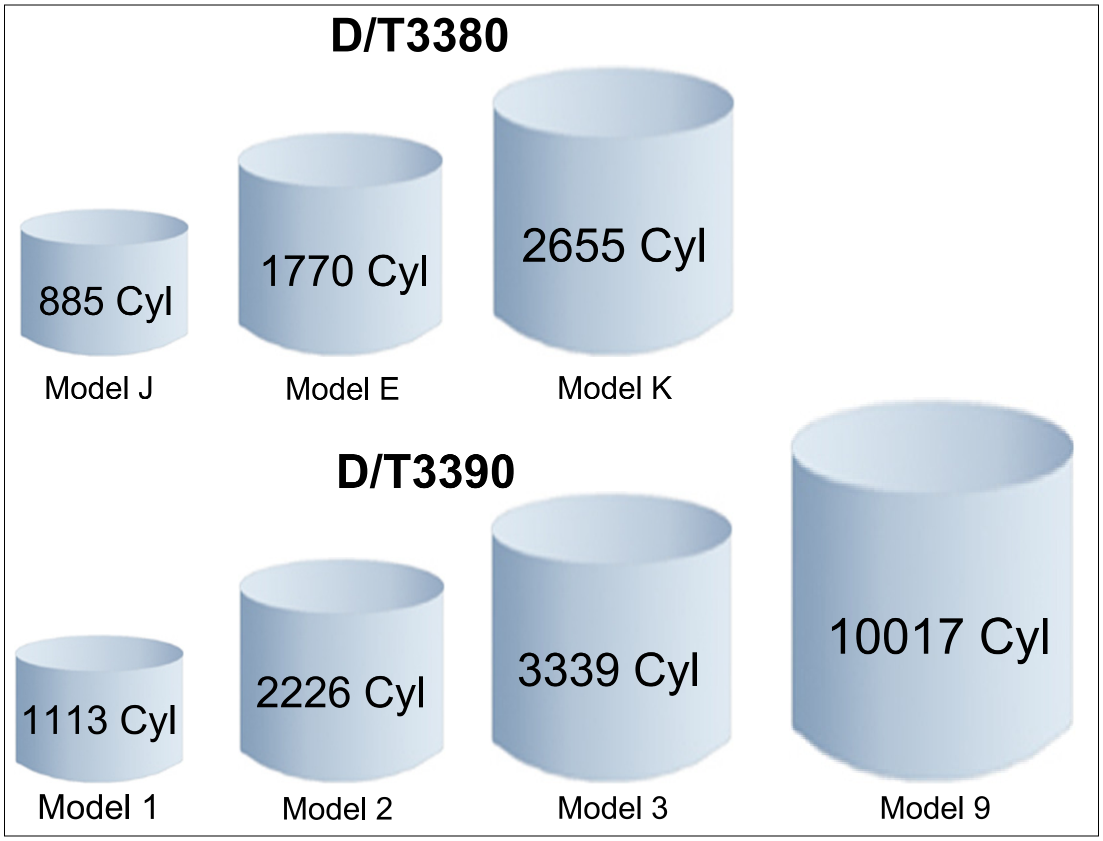


Figure 3-2 Traditional DASD capacity

DASD capacity

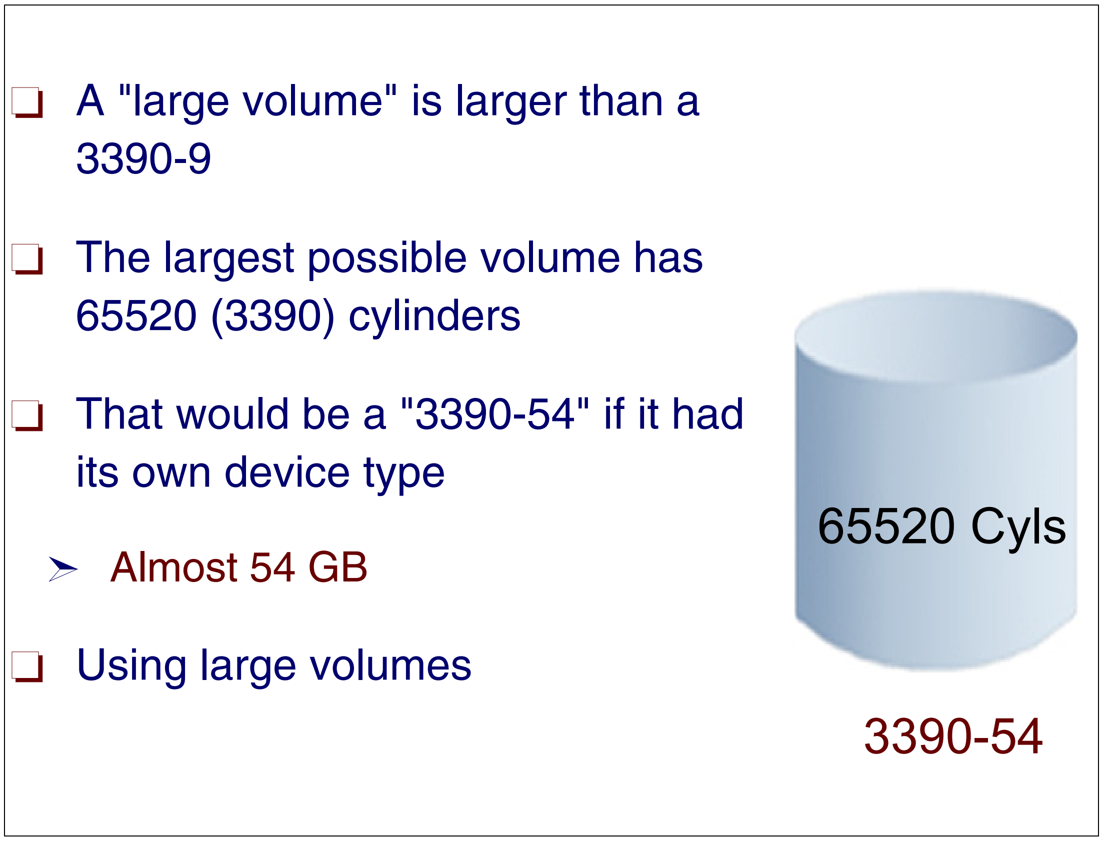
Figure 3-2 shows various DASD device types. 3380 devices were used in the 1980s. Capacity went from 885 to 2,655 cylinders per volume. When storage density increased, new device types were introduced at the end of the 1980s. Those types were called 3390. Capacity per volume ranged from 1,113 to 3,339 cylinders. A special device type, model 3390-9 was introduced to store large amounts of data that needed very fast access. The track geometry within one device category was (and is) always the same; this means that 3380 volumes have 47,476 bytes per track, and 3390 volumes have 56,664 bytes per track.

Table 3-1 lists further information about DASD capacity.

Table 3-1 DASD capacity

Physical characteristics	3380-J	3380-E	3380-K	3390-1	3390-2	3390-3	3390-9
Data Cyl/Device	855	1770	2655	1113	2226	3339	10017
Track/Cyl	15	15	15	15	15	15	15
Bytes/Trk	47476	47476	47476	56664	56664	56664	56664
Bytes/Cylinder	712140	712140	712140	849960	849960	849960	849960
MB/Device	630	1260	1890	946	1892	2838	8514

3.3 Large volume support



- ❑ A "large volume" is larger than a 3390-9
- ❑ The largest possible volume has 65520 (3390) cylinders
- ❑ That would be a "3390-54" if it had its own device type
  - ➢ Almost 54 GB
- ❑ Using large volumes

65520 Cyls
3390-54

Figure 3-3 Large volume support 3390-54

Large volume 3390-54

The IBM TotalStorage Enterprise Storage Server (ESS) initially supported custom volumes of up to 10017 cylinders, the size of the largest standard volume, the 3390 model 9. This was the limit set by the operating system software. The IBM TotalStorage ESS large volume support enhancement, announced in November 2001, has now increased the upper limit to 65520 cylinders, approximately 54 GB. The enhancement is provided as a combination of IBM TotalStorage ESS licensed internal code (LIC) changes and system software changes, available for z/OS and z/VM®.

Today, disk storage subsystems other than those listed can emulate one of those listed. For example, the IBM Enterprise Storage Server emulates the IBM 3390. On an emulated disk or on a VM minidisk, the number of cylinders per volume is a configuration option. It might be less than or greater than the stated number. If so, the number of bytes per device will differ accordingly. The IBM ESS Model 1750 supports up to 32760 cylinders and the IBM ESS Model 2107 supports up to 65520 cylinders.

Large volume support is available on z/OS operating systems, the ICKDSF, and DFSORT™ utilities.

Large volume support must be installed on all systems in a sysplex prior to sharing data sets on large volumes. Shared system and application data sets cannot be placed on large volumes until all system images in a sysplex have large volume support installed.

Large volume support design considerations

Benefits of using large volumes can be briefly summarized as follows:

- They reduce storage management tasks by allowing you to define and manage smaller configurations.
- They reduce the number of multivolume data sets you have to manage.
- They relieve architectural constraints by allowing you to address more data within the existing 64K subchannel number limit.

The size of the logical volume defined does not have an impact on the performance of the ESS subsystem. The ESS does not serialize I/O on the basis of logical devices, so an increase in the logical volume size does not affect the ESS backend performance. Host operating systems, on the other hand, serialize I/Os against devices. As more data sets reside on a single volume, there will be greater I/O contention accessing the device. With large volume support, it is more important than ever to try to minimize contention on the logical device level. To avoid potential I/O bottlenecks on devices:

- Exploit the use of Parallel Access Volumes to reduce IOS queuing on the system level.
- Eliminate unnecessary reserves by using WLM in goal mode.
- Multiple allegiance will automatically reduce queuing on sharing systems.

Parallel Access Volume (PAV) support is of key importance when implementing large volumes. PAV enables one MVS system to initiate multiple I/Os to a device concurrently. This keeps IOSQ times down and performance up even with many active data sets on the same volume. PAV is a practical “must” with large volumes. We discourage you from using large volumes without PAV. In particular, we recommend the use of dynamic PAV.

As the volume sizes grow larger, more data and data sets will reside on a single S/390 device address. Thus, the larger the volume, the greater the multi-system performance impact will be of serializing volumes with RESERVE processing. You need to exploit a GRS Star Configuration and convert all RESERVE's possible into system ENQ requests.

3.4 Redundant array of independent disks (RAID)

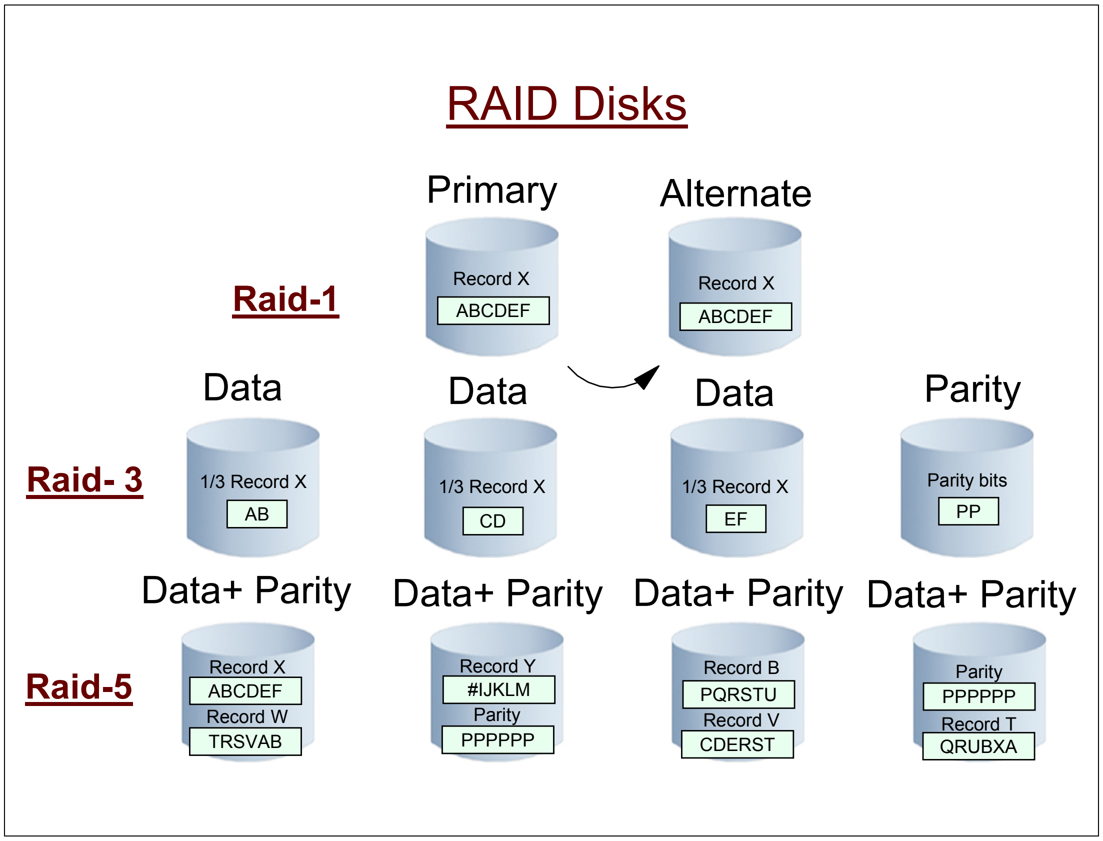


Figure 3-4 Redundant array of independent disks (RAID)

RAID architecture

Redundant array of independent disks (RAID) is a direct access storage architecture where data is recorded across multiple physical disks with parity separately recorded, so that no loss of access to data results from the loss of any one disk in the array.

RAID breaks the one-to-one association of volumes with devices. A logical volume is now the addressable entity presented by the controller to the attached systems. The RAID unit maps the logical volume across multiple physical devices. Similarly, blocks of storage on a single physical device may be associated with multiple logical volumes. Because a logical volume is mapped by the RAID unit across multiple physical devices, it is now possible to overlap processing for multiple cache misses to the same logical volume because cache misses can be satisfied by different physical devices.

The RAID concept involves many small computer system interface (SCSI) disks replacing a big one. The major RAID advantages are:

- Performance (due to parallelism)
- Cost (SCSI are commodities)
- zSeries compatibility
- Environment (space and energy)

However, RAID increased the chances of malfunction due to media and disk failures and the fact that the logical device is now residing on many physical disks. The solution was redundancy, which wastes space and causes performance problems as “write penalty” and “free space reclamation.” To address this performance issue, large caches are implemented.

Note: The ESS storage controllers use the RAID architecture that enables multiple logical volumes to be mapped on a single physical RAID group. If required, you can still separate data sets on a physical controller boundary for the purpose of availability.

RAID implementations

Except for RAID-1, each manufacturer sets the number of disks in an array. An *array* is a set of logically related disks, where a parity applies.

Various implementations certified by the RAID Architecture Board are:

RAID-1 This has just disk mirroring, like dual copy.

RAID-3 This has an array with one dedicated parity disk and just one I/O request at a time, with intra-record striping. It means that the written physical block is striped and each piece (together with the parity) is written in parallel in each disk of the array. The access arms move together. It has a high data rate and a low I/O rate.

RAID-5 This has an array with one distributed parity (there is no dedicated disk for parities). It does I/O requests in parallel with extra-record striping, meaning each physical block is written in each disk. The access arms move independently. It has strong caching to avoid write penalties; that is, four disk I/Os per write. RAID-5 has a high I/O rate and a medium data rate. RAID-5 is used by the IBM 2105 controller with 8-disk arrays in the majority of configurations.

RAID-5 does the following:

- It reads data from an undamaged disk. This is just one, single disk I/O operation.
- It reads data from a damaged disk, which implies (n-1) disk I/Os, to recreate the lost data where n is the number of disks in the array.
- For every write to an undamaged disk, RAID-5 does four I/O operations in order to store a correct parity block; this is called a write penalty. This penalty can be relieved with strong caching and a slice triggered algorithm (coalescing disks updates from cache into a single parallel I/O).
- For every write to a damaged disk, RAID-5 does n-1 reads and one parity write.

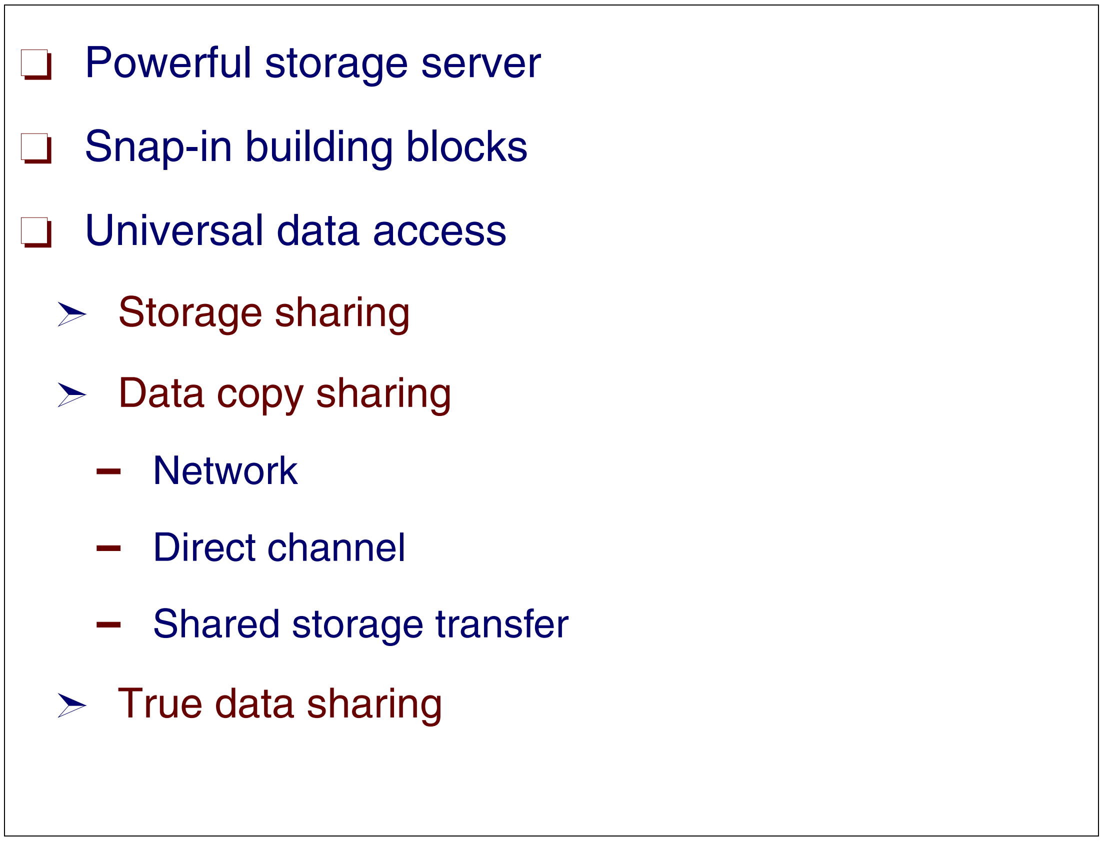
RAID-6 This has an array with two distributed parity and I/O requests in parallel with extra-record striping. Its access arms move independently (Reed/Salomon P-Q parity). The write penalty is greater than RAID-5, with six I/Os per write.

RAID-6+ This is without write penalty (due to log-structured file, or LFS), and has background free-space reclamation. The access arms all move together for writes. It is used by the RVA controller.

RAID-10 RAID-10 has a new RAID architecture designed to give performance for striping and has redundancy for mirroring. RAID-10 is optionally implemented in the IBM 2105.

Note: Data striping (stripe sequential physical blocks in different disks) is sometimes called RAID-0, but it is not a real RAID because of no redundancy, that is, no parity bits.

3.5 Seascape architecture



- Powerful storage server
- Snap-in building blocks
- Universal data access
  - Storage sharing
  - Data copy sharing
    - Network
    - Direct channel
    - Shared storage transfer
  - True data sharing

Figure 3-5 Seascape architecture

Seascape architecture

The IBM Enterprise Storage Server's "architecture for e-business" design is based on IBM's storage enterprise architecture, Seascape. The Seascape architecture defines next-generation concepts for storage by integrating modular building block technologies from IBM, including disk, tape, and optical storage media, powerful processors, and rich software. Integrated Seascape solutions are highly reliable, scalable, and versatile, and support specialized applications on servers ranging from PCs to super computers. Virtually all types of servers can concurrently attach to the ESS, including iSeries® and AS/400® systems. As a result, ESS can be the external disk storage system of choice for AS/400 as well as iSeries systems in heterogeneous SAN environments.

Seascape has three basic concepts:

- Powerful storage server
- Snap-in building blocks
- Universal data access

DFSMS provides device support for the IBM 2105 Enterprise Storage Server (ESS), a high-end storage subsystem. The ESS storage subsystem succeeded the 3880, 3990, and 9340 subsystem families. Designed for mid-range and high-end environments, the ESS gives you large capacity, high performance, continuous availability, and storage expandability. You can read more about ESS in "Enterprise Storage Server (ESS)" on page 54.

The ESS was the first of the Seascape architecture storage products to attach directly to IBM System z and open system platforms. The Seascape architecture products come with integrated storage controllers. These integrated storage controllers allow the attachment of physical storage devices that emulate 3390 Models 2, 3, and 9, or provide 3380 track compatibility mode.

Powerful storage server

The storage system is intelligent and independent, and it can be reached by channels or via the network. It is powered by a set of fast RISC processors.

Snap-in building blocks

Each Seascape product is comprised of building blocks, such as:

- Scalable n-way RISC server, PCI-based. This provides the logic of the storage server.
- Memory cache from RISC processor memory.
- Channel attachments, such as FC-AL, SCSI, ESCON, FICON® and SSA.
- Network attachments, such as Ethernet, FDDI, TR, and ATM.
- These attachments may also implement functions—a mix of network interfaces (to be used as a remote and independent storage server) and channel interfaces (to be used as a storage controller interface).
- Software building blocks, such as an AIX® subset, Java™ applications, and Tivoli® Storage Manager. High level language (HLL) is more flexible than microcode, and is easier to write and maintain.
- Storage adapters, for mixed storage devices technologies.
- Storage device building blocks, such as serial disk (7133), 3590 tape (Magstar), and optical (3995).
- Silos and robots (3494).

Universal data access

Universal data access allows a wide array of connectivity, such as z/OS, UNIX, Linux®, and OS/400®, to common data. There are three types of universal access:

- **Storage sharing**

  Physical storage (DASD or tape) is statically divided into fixed partitions available to a given processor. It is not a software function. The subsystem controller knows which processors own which storage partitions. In a sense, only capacity is shared, not data; one server cannot access the data of the other server. It is required that the manual reassignment of storage capacity between partitions be simple and nondisruptive.

  The advantages are:

  – Purchase higher quantities with greater discounts
  – Only one type of storage to manage
  – Static shifting of capacity as needed

  The drawbacks are:

  – Higher price for SCSI data
  – Collocation at 20 meters of the SCSI servers
  – No priority concept between z/OS and UNIX/NT I/O requests

- **Data copy sharing**

  Data copy sharing is an interim data replication solution (waiting for a true data sharing) done via data replication from one volume accessed by a platform to another volume accessed by another platform. The replication can be done through software or hardware.There are three ways to implement data copy sharing:

  - Network: Via network data transfer, such as SNA or TCP/IP. This method has drawbacks, such as CPU and network overhead; it is still slow and expensive for massive data transfer.
  - Direct channel: Direct data transfer between the processors involved using channel or bus capabilities, referred to as *bulk data transfer*.
  - Shared storage transfer: Writing an intermediate flat file by software into the storage subsystem cache, that is read (and translated) by the receiving processor, so the storage is shared.

- **True data sharing**

  For data sharing between multiple platforms for read/write of a single copy that addresses the complex issues of mixed data types, file structures, databases, and SCPs, there is no available solution.

3.6 Enterprise Storage Server (ESS)

- Two 6-way RISC processors (668 MHZ)
- 4.8 GB/sec of aggregate bandwidth
- Up to 32 ESCON / SCSI / mixed
- Up to 16 FICON and FCP channels
- Up to 64 GB of cache
- 2 GB of NVS cache
- 18.2 / 36.4 / 72.8 / 145.6 GB capacity disk options
- Up to 55.9 TB capacity
- 8 x 160 MB/sec SSA loops
- 10,000 rpm and 15,000 rpm disk options
- Connects to SAN
- RAID-5 or RAID-10

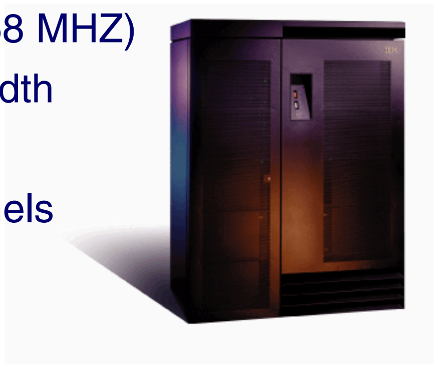


Figure 3-6 Enterprise Storage Server model 800

Enterprise Storage Server (ESS)

The IBM Enterprise Storage Server (ESS) is a high performance, high availability capacity storage subsystem. It contains two six-way RISC processors (668 MHZ) with up to 64 GB cache and 2 GB of non-volatile storage (NVS) to protect from data loss during power outages. Connectivity to IBM mainframes is through up to 32 ESCON channels and up to 16 FICON channels. For other platforms, such as IBM System i™, UNIX, or NT, the connectivity is through up to 32 SCSI interfaces.

Cache

Cache is used to store both read and write data to improve ESS performance to the attached host systems. There is the choice of 8, 16, 24, 32, or 64 GB of cache. This cache is divided between the two clusters of the ESS, giving the clusters their own non-shared cache. The ESS cache uses ECC (error checking and correcting) memory technology to enhance reliability and error correction of the cache. ECC technology can detect single- and double-bit errors and correct all single-bit errors. Memory scrubbing, a built-in hardware function, is also performed and is a continuous background read of data from memory to check for correctable errors. Correctable errors are corrected and rewritten to cache. To protect against loss of data on a write operation, the ESS stores two copies of written data, one in cache and the other in NVS.

NVS cache

NVS is used to store a second copy of write data to ensure data integrity, should there be a power failure or a cluster failure and the cache copy is lost. The NVS of cluster 1 is located in cluster 2 and the NVS of cluster 2 is located in cluster 1. In this way, in the event of a cluster failure, the write data for the failed cluster will be in the NVS of the surviving cluster. This write data is then de-staged at high priority to the disk arrays. At the same time, the surviving cluster will start to use its own NVS for write data, ensuring that two copies of write data are still maintained. This ensures that no data is lost even in the event of a component failure.

ESS Model 800

The ESS Model 800 has a 2 GB NVS. Each cluster has 1 GB of NVS, made up of four cards. Each pair of NVS cards has its own battery-powered charger system that protects data even if power is lost on the entire ESS for up to 72 hours. This model has the following enhancements:

- Model 800 allows 4.8 GB/sec of aggregate bandwidth.
- In the disk interface the ESS has eight Serial Storage Architecture (SSA) loops, each one with a rate of 160 MB/sec for accessing the disks. See “SSA loops” on page 65 for more information about this topic.
- ESS implements RAID-5 or RAID-10 for availability and has eight disks in the majority of the arrays. See “RAID-10” on page 67 for more information about this topic.
- Four sizes disks of 18.2, 36.4, 72.8, and 145.6 GB, which can be intermixed. The ESS maximum capacity is over 55.9 TB with a second frame attached.

ESS Model 750

The ESS 750 is intended for smaller enterprises that need the enterprise-level advanced functions, reliability, and availability offered by the Model 800, but at a lower entry cost and size. It is specifically designed to meet the high demands of medium-sized mainframe environments, and for this reason it is closely tied in with IBM's z890 offering.

The ESS 750 has capabilities similar to the ESS 800. The ESS Model 750 consists of two clusters, each with a two-way processor and 4 or 8 GB cache. It can have two to six Fibre Channel/FICON or ESCON host adapters. The storage capacity ranges from a minimum of 1.1 TB up to a maximum of 4 TB. A key feature is that the ESS 750 is upgradeable, non-disruptively, to the ESS Model 800, which can grow to more than 55 TB of physical capacity.

Note: Effective April 28, 2006, IBM withdrew from marketing the following products:

- IBM TotalStorage Enterprise Storage Server (ESS) Models 750 and 800
- IBM Standby Capacity on Demand for ESS offering

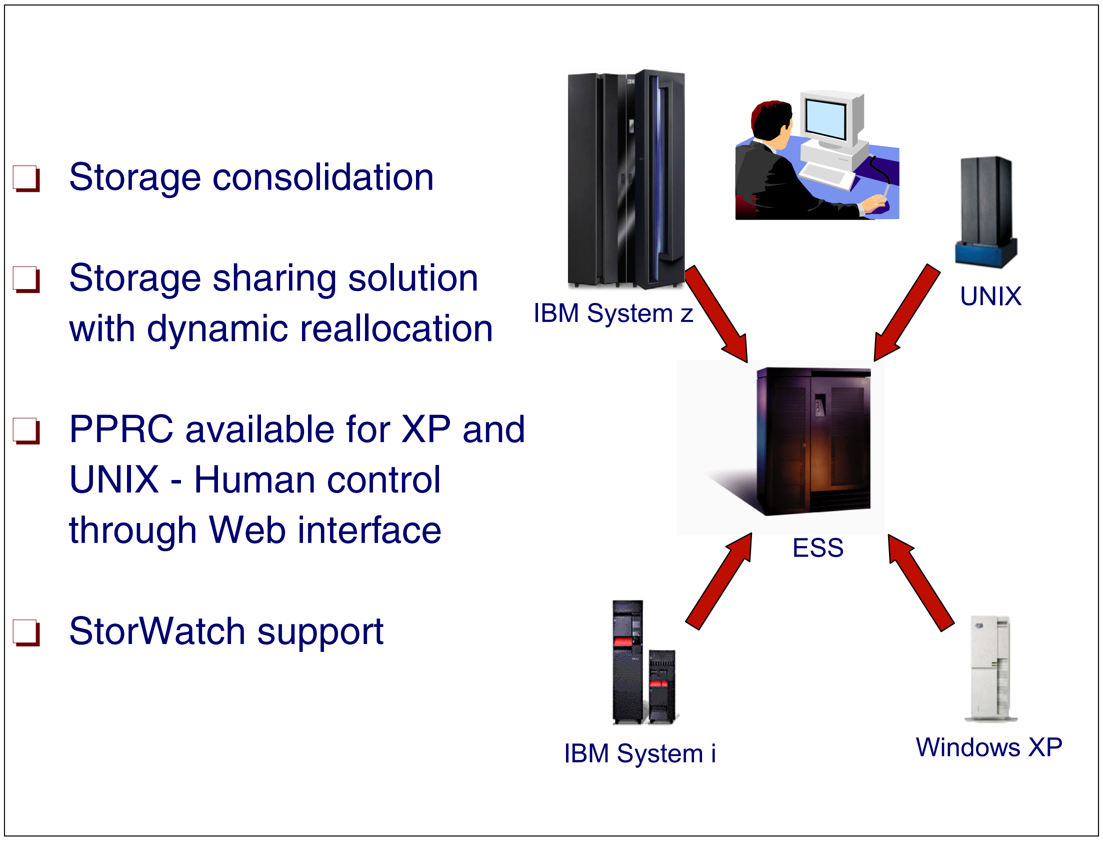
For replacement products, see “IBM TotalStorage DS6000” on page 82 and “IBM TotalStorage DS8000” on page 85.

SCSI protocol

Although we do not cover other platforms in this publication, we provide here a brief overview of the SCSI protocol. The SCSI adapter is a card in the host. It connects to a SCSI bus via a SCSI port. There are two different types of SCSI supported by ESS:

- SCSI Fast Wide with 20 MB/sec
- Ultra™ SCSI Wide with 40 MB/sec

3.7 ESS universal access



- Storage consolidation
- Storage sharing solution with dynamic reallocation
- PPRC available for XP and UNIX - Human control through Web interface
- StorWatch support

Figure 3-7 ESS universal access

ESS universal access

ESS is a product designed to implement *storage* consolidation that puts all of your enterprise data under the same cover. This consolidation is the first step in achieving *server* consolidation – that is, to put all of your enterprise applications under the same z/OS cluster.

PPRC support

IBM includes a Web browser interface called TotalStorage Enterprise Storage Server (ESS) Copy Services. The interface is part of the ESS subsystem and can be used to perform FlashCopy and PPRC functions.

Many of the ESS features are now available to non-zSeries platforms, such as PPRC for Windows® XP and UNIX, where the control is through a Web interface.

StorWatch support

On the software side, there is StorWatch, a range of products in UNIX/XP that does what DFSMS and automation do for System z. The TotalStorage Expert, formerly marketed as StorWatch Expert, is a member of the IBM and Tivoli Systems family of solutions for Enterprise Storage Resource Management (ESRM). These are offerings that are designed to complement one another, and provide a total storage management solution.

TotalStorage Expert is an innovative software tool that gives administrators powerful, yet flexible storage asset, capacity, and performance management capabilities to centrally manage Enterprise Storage Servers located anywhere in the enterprise.

3.8 ESS major components

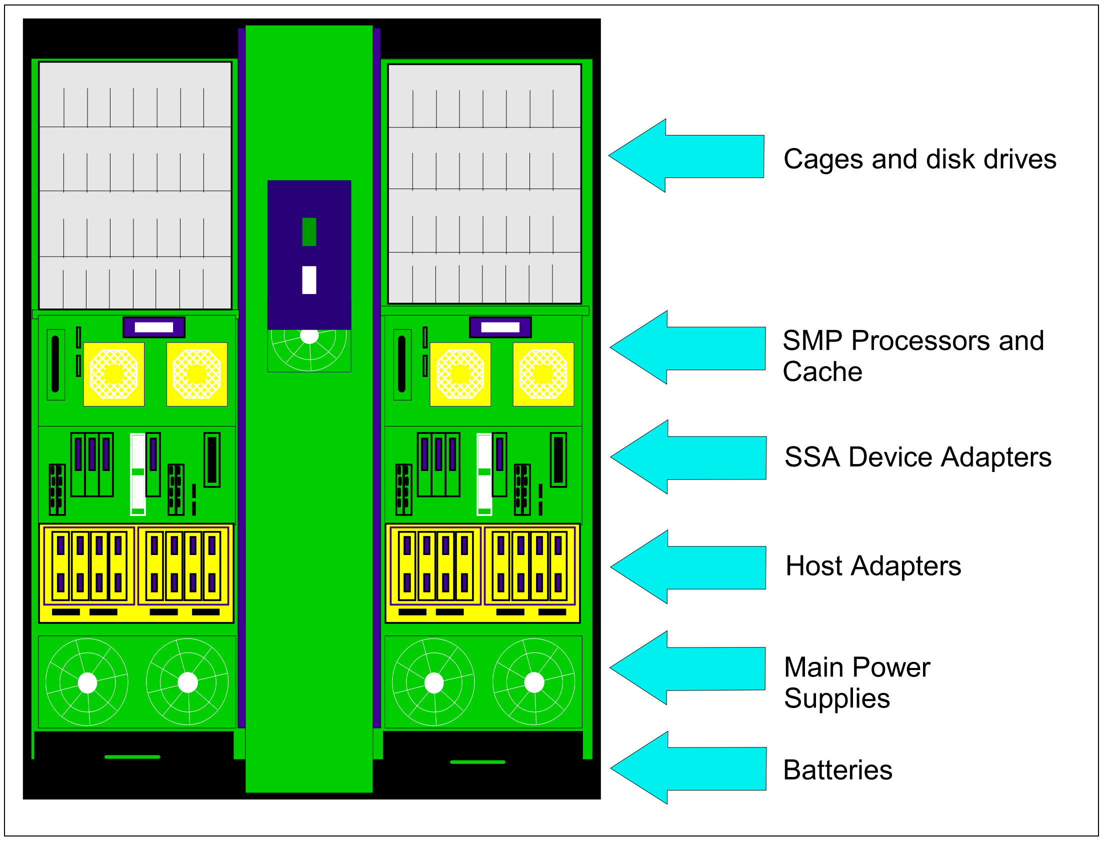


Figure 3-8 ESS major components

ESS Model 800 major components

Figure 3-8 shows an IBM TotalStorage Enterprise Storage Server Model 800 and its major components. As you can see, the ESS base rack consists of two clusters, each with its own power supplies, batteries, SSA device adapters, processors, cache and NVS, CD drive, hard disk, floppy disk, and network connections. Both clusters have access to any host adapter card, even though they are physically spread across the clusters.

At the top of each cluster is an ESS cage. Each cage provides slots for up to 64 disk drives, 32 in front and 32 at the back.

This storage box has two enclosures:

- A base enclosure, with:
  - Two 3-phase power supplies
  - Up to 128 disk drives in two cages
  - Feature #2110 for Expansion Enclosure attachment
  - Host adapters, cluster processors, cache and NVS, and SSA device adapters
- An expansion enclosure, with:
  - Two 3-phase power supplies
  - Up to 256 disk drives in four cages for additional capacity

3.9 ESS host adapters

Host adapter bays

- ❑ 4 bays
- ❑ 4 host adapters per bay

ESCON host adapters

- ❑ Up to 32 ESCON links
- ❑ 2 ESCON links per host adapter

2 Gb FICON host adapters

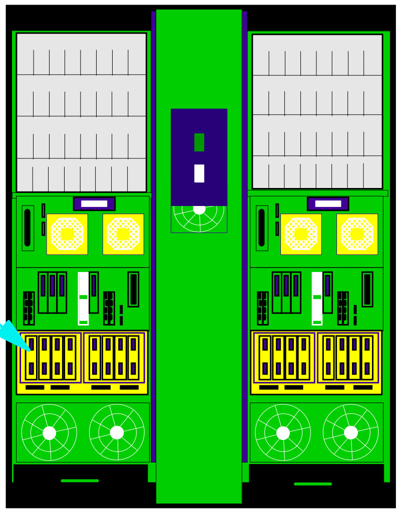
- ❑ Up to 16 FICON links
- ❑ 1 FICON link per host adapter
- ❑ Auto speed detection - 1 Gb or 2 Gb

SCSI host adapters

- ❑ Up to 32 SCSI bus connections
- ❑ 2 SCSI ports per host adapter

2 Gb Fibre Channel host adapters

- ❑ Up to 16 Fibre Channel links
- ❑ 1 Fibre Channel port per host adapter
- ❑ Auto speed detection - 1 Gb or 2 Gb



Adapters can be intermixed

- ❑ Any combination of host adapter cards up to a maximum of 16

Figure 3-9 Host adapters

ESS host adapters

The ESS has four host adapter (HA) bays, two in each cluster. Each bay supports up to four host adapter cards. Each of these host adapter cards can be for FICON, ESCON, SCSI, or Fibre Channel server connection. Figure 3-9 lists the main characteristics of the ESS host adapters.

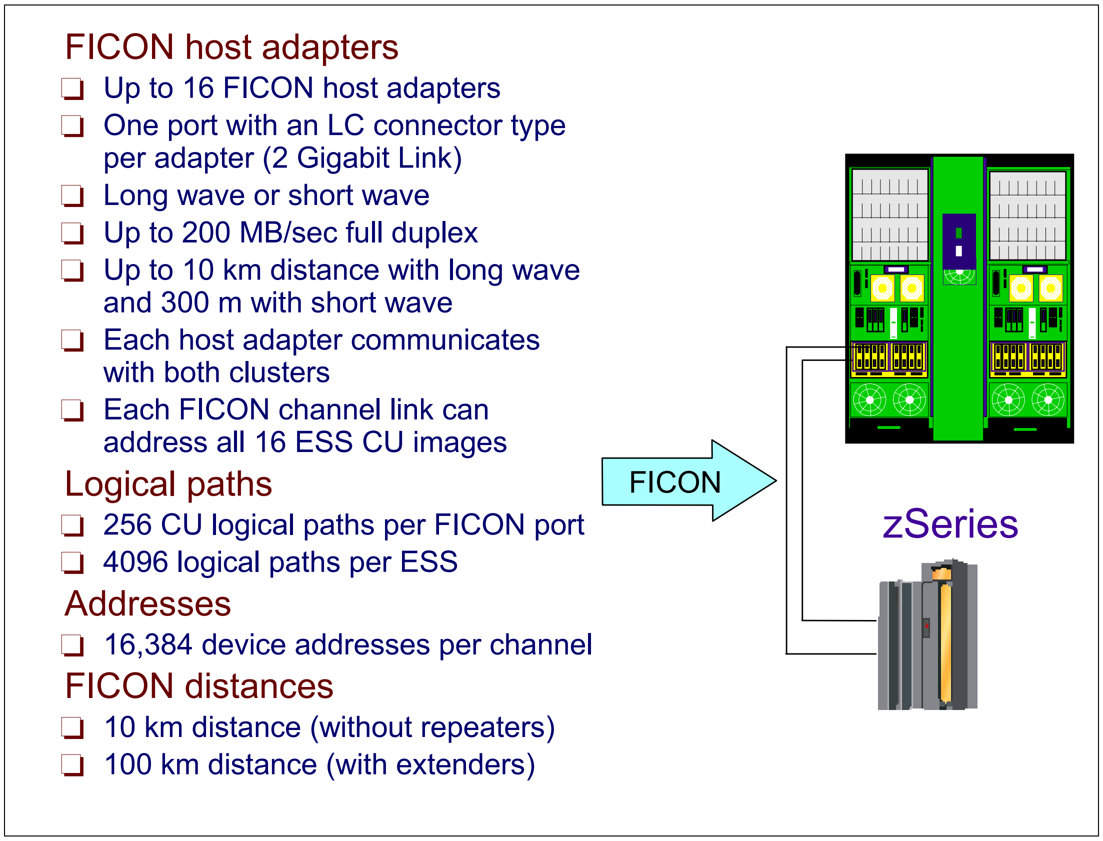
Each host adapter can communicate with either cluster. To install a new host adapter card, the bay must be powered off. For the highest path availability, it is important to spread the host connections across all the adapter bays. For example, if you have four ESCON links to a host, each connected to a different bay, then the loss of a bay for upgrade would only impact one out of four of the connections to the server. The same would be valid for a host with FICON connections to the ESS.

Similar considerations apply for servers connecting to the ESS by means of SCSI or fibre channel links. For open system servers, the Subsystem Device Driver (SDD) program that comes standard with the ESS can be installed on the connecting host servers to provide multiple paths or connections to handle errors (path failover) and balance the I/O load to the ESS.

The ESS connects to a large number of different servers, operating systems, host adapters, and SAN fabrics. A complete and current list is available at the following Web site:

`http://www.storage.ibm.com/hardsoft/products/ess/supserver.htm`

3.10 FICON host adapters



FICON host adapters

- ❑ Up to 16 FICON host adapters
- ❑ One port with an LC connector type per adapter (2 Gigabit Link)
- ❑ Long wave or short wave
- ❑ Up to 200 MB/sec full duplex
- ❑ Up to 10 km distance with long wave and 300 m with short wave
- ❑ Each host adapter communicates with both clusters
- ❑ Each FICON channel link can address all 16 ESS CU images

Logical paths

- ❑ 256 CU logical paths per FICON port
- ❑ 4096 logical paths per ESS

Addresses

- ❑ 16,384 device addresses per channel

FICON distances

- ❑ 10 km distance (without repeaters)
- ❑ 100 km distance (with extenders)

Figure 3-10 FICON host adapter

FICON host adapters

FICON (Fiber Connection) is based on the standard Fibre Channel architecture, and therefore shares the attributes associated with Fibre Channel. This includes the common FC-0, FC-1, and FC-2 architectural layers, the 100 MBps bidirectional (full-duplex) data transfer rate, and the point-to-point distance capability of 10 kilometers. The ESCON protocols have been mapped to the FC-4 layer, the Upper Level Protocol (ULP) layer, of the Fibre Channel architecture. All this provides a full-compatibility interface with previous S/390 software and puts the zSeries servers in the Fibre Channel industry standard.

FICON versus ESCON

FICON goes beyond ESCON limits:

- Addressing limit, from 1024 device addresses per channel to up to 16,384 (maximum of 4096 devices supported within one ESS).
- Up to 256 control unit logical paths per port.
- FICON channel to ESS allows multiple concurrent I/O connections (the ESCON channel supports only one I/O connection at one time).
- Greater channel and link bandwidth: FICON has up to 10 times the link bandwidth of ESCON (1 Gbps full-duplex, compared to 200 MBps half duplex). FICON has up to more than four times the effective channel bandwidth.
- FICON path consolidation using switched point-to-point topology.

- Greater unrepeated fiber link distances (from 3 km for ESCON to up to 10 km, or 20 km with an RPQ, for FICON).

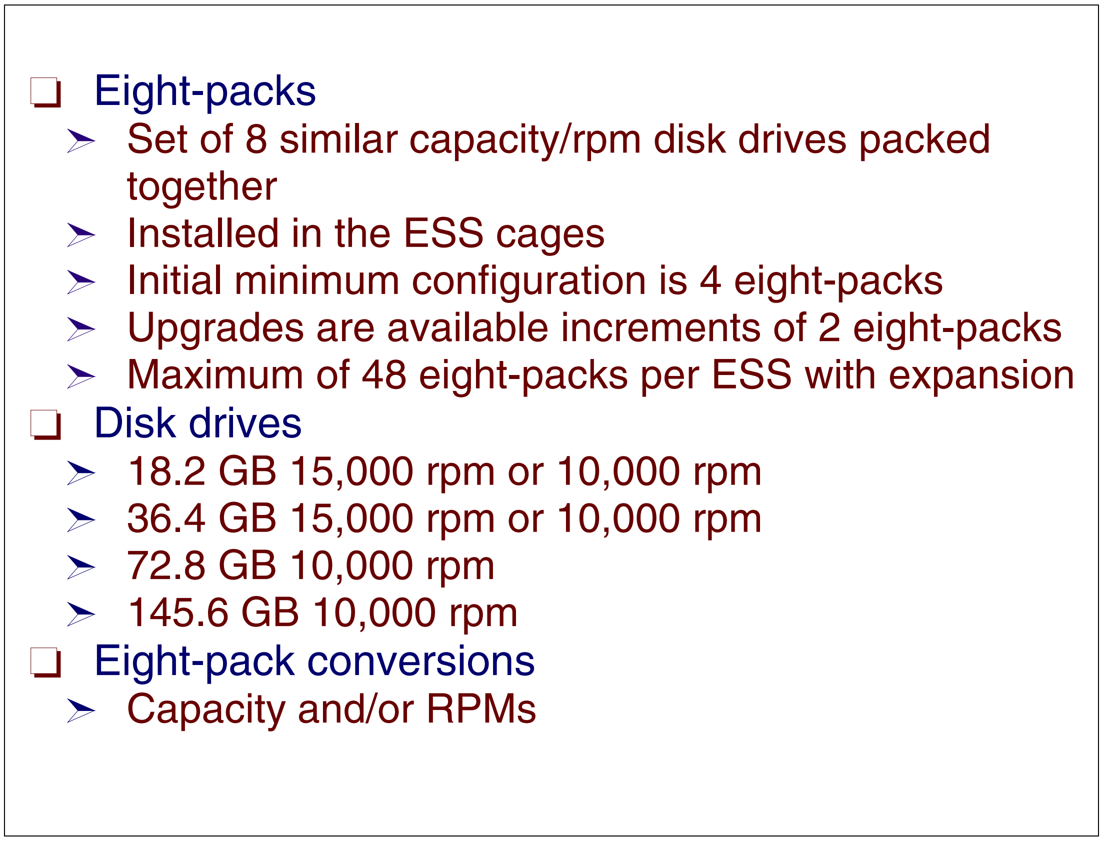
These characteristics allow simpler and more powerful configurations. The ESS supports up to 16 host adapters, which allows for a maximum of 16 Fibre Channel/FICON ports per machine, as shown in Figure 3-10.

Each Fibre Channel/FICON host adapter provides one port with an LC connector type. The adapter is a 2 Gb card and provides a nominal 200 MBps full-duplex data rate. The adapter will auto-negotiate between 1 Gb and 2 Gb, depending upon the speed of the connection at the other end of the link. For example, from the ESS to a switch/director, the FICON adapter can negotiate to 2 Gb if the switch/director also has 2 Gb support. The switch/director to host link can then negotiate at 1 Gb.

Host adapter cards

There are two types of host adapter cards you can select: long wave (feature 3024), and short wave (feature 3025). With long-wave laser, you can connect nodes at distances of up to 10 km (without repeaters). With shortwave laser, you can connect at distances of up to 300 m. These distances can be extended using switches/directors.

3.11 ESS disks



- ❑ Eight-packs
  - ➢ Set of 8 similar capacity/rpm disk drives packed together
  - ➢ Installed in the ESS cages
  - ➢ Initial minimum configuration is 4 eight-packs
  - ➢ Upgrades are available increments of 2 eight-packs
  - ➢ Maximum of 48 eight-packs per ESS with expansion
- ❑ Disk drives
  - ➢ 18.2 GB 15,000 rpm or 10,000 rpm
  - ➢ 36.4 GB 15,000 rpm or 10,000 rpm
  - ➢ 72.8 GB 10,000 rpm
  - ➢ 145.6 GB 10,000 rpm
- ❑ Eight-pack conversions
  - ➢ Capacity and/or RPMs

Figure 3-11 ESS disks

ESS disks

With a number of disk drive sizes and speeds available, including intermix support, the ESS provides a great number of capacity configuration options.

The maximum number of disk drives supported within the IBM TotalStorage Enterprise Storage Server Model 800 is 384—with 128 disk drives in the base enclosure and 256 disk drives in the expansion rack. When configured with 145.6 GB disk drives, this gives a total physical disk capacity of approximately 55.9 TB (see Table 3-2 for more details).

Disk drives

The minimum available configuration of the ESS Model 800 is 582 GB. This capacity can be configured with 32 disk drives of 18.2 GB contained in four eight-packs, using one ESS cage. All incremental upgrades are ordered and installed in pairs of eight-packs; thus the minimum capacity increment is a pair of similar eight-packs of either 18.2 GB, 36.4 GB, 72.8 GB, or 145.6 GB capacity.

The ESS is designed to deliver substantial protection against data corruption, not just relying on the RAID implementation alone. The disk drives installed in the ESS are the latest state-of-the-art magneto resistive head technology disk drives that support advanced disk functions such as disk error correction codes (ECC), Metadata checks, disk scrubbing, and predictive failure analysis.

Eight-pack conversions

The ESS eight-pack is the basic unit of capacity within the ESS base and expansion racks. As mentioned before, these eight-packs are ordered and installed in pairs. Each eight-pack can be configured as a RAID 5 rank (6+P+S or 7+P) or as a RAID 10 rank (3+3+2S or 4+4).

The IBM TotalStorage ESS Specialist will configure the eight-packs on a loop with spare DDMs as required. Configurations that include drive size intermixing may result in the creation of additional DDM spares on a loop as compared to non-intermixed configurations. Currently there is the choice of four different new-generation disk drive capacities for use within an eight-pack:

- 18.2 GB/15,000 rpm disks
- 36.4 GB/15,000 rpm disks
- 72.8 GB/10,000 rpm disks
- 145.6 GB/10,000 rpm disks

Also available is the option to install eight-packs with:

- 18.2 GB/10,000 rpm disks or
- 36.4 GB/10,000 rpm disks

The eight disk drives assembled in each eight-pack are all of the same capacity. Each disk drive uses the 40 MBps SSA interface on each of the four connections to the loop.

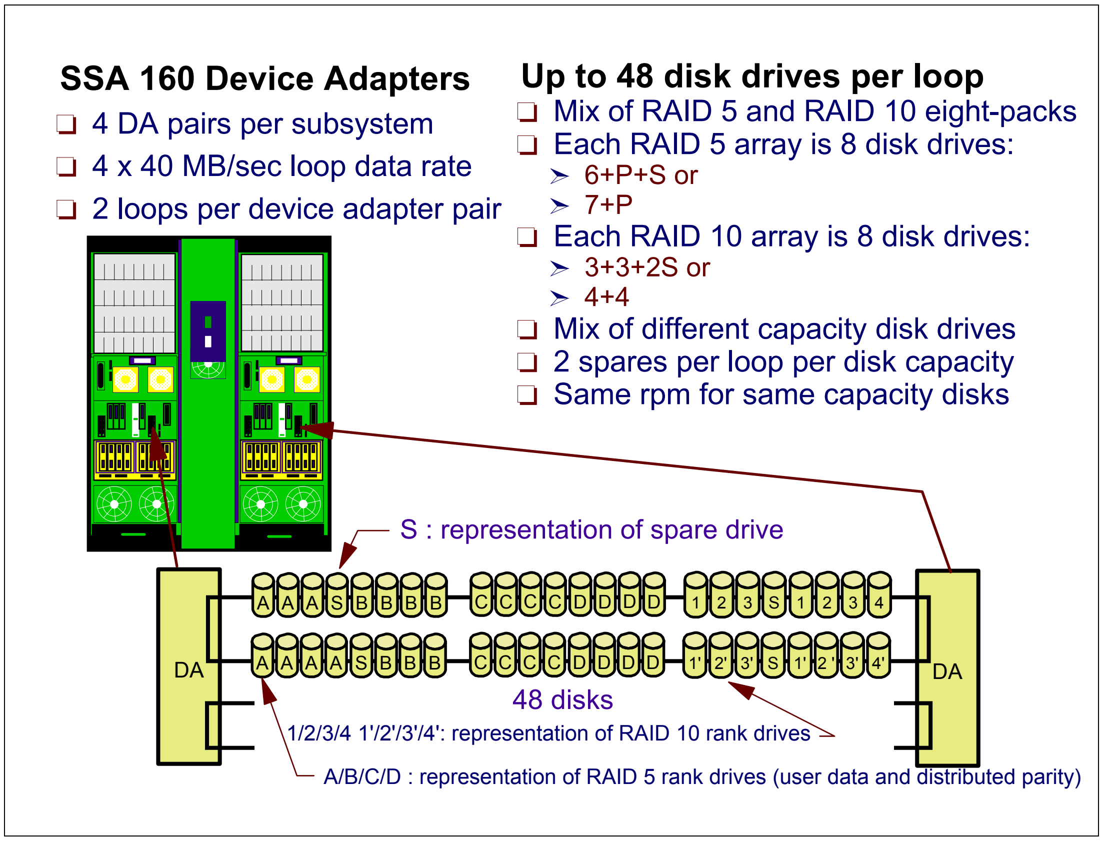
It is possible to mix eight-packs of different capacity disks and speeds (rpm) within an ESS, as described in the following sections.

Table 3-2 should be used as a guide for determining the capacity of a given eight-pack. This table shows the capacities of the disk eight-packs when configured as RAID ranks. These capacities are the *effective capacities* available for user data.

Table 3-2 Disk eight-pack effective capacity chart (gigabytes)

Disk size	**Physical capacity (raw capacity)**	**Effective usable capacity (2)**			
		RAID 10		**RAID 5 (3)**	
		3 + 3 + 2S Array (4)	**4 + 4 Array (5)**	**6+P+S Array (6)**	**7 + P Array (7)**
18.2	145.6	52.50	70.00	105.20	122.74
36.4	291.2	105.12	140.16	210.45	245.53
72.8	582.4	210.39	280.52	420.92	491.08
145.6	1,164.8	420.78	561.04	841.84	982.16

3.12 ESS device adapters



SSA 160 Device Adapters

- 4 DA pairs per subsystem
- 4 x 40 MB/sec loop data rate
- 2 loops per device adapter pair

Up to 48 disk drives per loop

- Mix of RAID 5 and RAID 10 eight-packs
- Each RAID 5 array is 8 disk drives:
  - 6+P+S or
  - 7+P
- Each RAID 10 array is 8 disk drives:
  - 3+3+2S or
  - 4+4
- Mix of different capacity disk drives
- 2 spares per loop per disk capacity
- Same rpm for same capacity disks

S : representation of spare drive
A A A S B B B B C C C C D D D D 1 2 3 S 1 2 3 4
A A A A S B B B C C C C D D D D 1' 2' 3' S 1' 2' 3' 4'
DA
DA
48 disks
1/2/3/4 1'/2'/3'/4': representation of RAID 10 rank drives
A/B/C/D : representation of RAID 5 rank drives (user data and distributed parity)

Figure 3-12 Device adapters

ESS device adapters

Device adapters (DA) provide the connection between the clusters and the disk drives. The ESS Model 800 implements faster Serial Storage Architecture (SSA) device adapters than its predecessor models.

The ESS Storage Server Model 800 uses the latest SSA160 technology in its device adapters (DA). With SSA 160, each of the four links operates at 40 MBps, giving a total nominal bandwidth of 160 MBps for each of the two connections to the loop. This amounts to a total of 320 MBps across each loop. Also, each device adapter card supports two independent SSA loops, giving a total bandwidth of 320 MBps per adapter card. There are eight adapter cards, giving a total nominal bandwidth capability of 2,560 MBps. Refer to “SSA loops” on page 65 for more information about this topic.

SSA loops

One adapter from each pair of adapters is installed in each cluster as shown in Figure 3-12. The SSA loops are between adapter pairs, which means that all the disks can be accessed by both clusters. During the configuration process, each RAID array is configured by the IBM TotalStorage ESS Specialist to be normally accessed by only one of the clusters. Should a cluster failure occur, the remaining cluster can take over all the disk drives on the loop.

RAID 5 and RAID 10

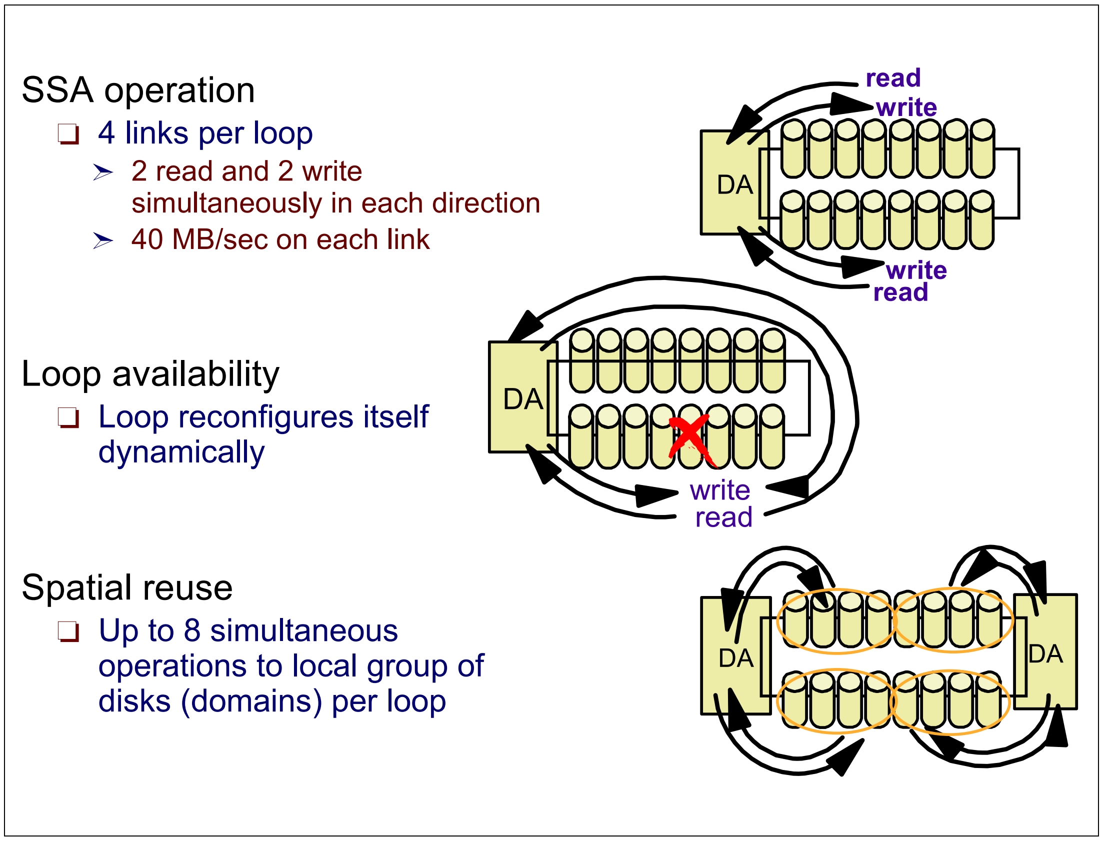
RAID 5 and RAID 10 are managed by the SSA device adapters. RAID 10 is explained in detail in “RAID-10” on page 67. Each loop supports up to 48 disk drives, and each adapter pair supports up to 96 disk drives. There are four adapter pairs supporting up to 384 disk drives in total. Figure 3-12 shows a logical representation of a single loop with 48 disk drives (RAID ranks are actually split across two eight-packs for optimum performance). You can see there are six RAID arrays: four RAID 5 designated A to D, and two RAID 10 (one 3+3+2 spare and one 4+4).

Disk drives per loop

Each loop supports up to 48 disk drives, and each adapter pair supports up to 96 disk drives. There are four adapter pairs supporting up to 384 disk drives in total.

Figure 3-12 on page 63 shows a logical representation of a single loop with 48 disk drives (RAID ranks are actually split across two eight-packs for optimum performance). In the figure you can see there are six RAID arrays: four RAID 5 designated A to D, and two RAID 10 (one 3+3+2 spare and one 4+4).

3.13 SSA loops


SSA operation

- ❑ 4 links per loop
  - ➢ 2 read and 2 write simultaneously in each direction
  - ➢ 40 MB/sec on each link

Loop availability

- ❑ Loop reconfigures itself dynamically

Spatial reuse

- ❑ Up to 8 simultaneous operations to local group of disks (domains) per loop

Figure 3-13 SSA loops

SSA operation

SSA is a high performance, serial connection technology for disk drives. SSA is a full-duplex loop-based architecture, with two physical read paths and two physical write paths to every disk attached to the loop. Data is sent from the adapter card to the first disk on the loop and then passed around the loop by the disks until it arrives at the target disk. Unlike bus-based designs, which reserve the whole bus for data transfer, SSA only uses the part of the loop between adjacent disks for data transfer. This means that many simultaneous data transfers can take place on an SSA loop, and it is one of the main reasons that SSA performs so much better than SCSI. This simultaneous transfer capability is known as “spatial release.”

Each read or write path on the loop operates at 40 MB/s, providing a total loop bandwidth of 160 MB/s.

Loop availability

The loop is a self-configuring, self-repairing design that allows genuine hot-plugging. If the loop breaks for any reason, then the adapter card will automatically reconfigure the loop into two single loops. In the ESS, the most likely scenario for a broken loop is if the actual disk drive interface electronics should fail. If this should happen, the adapter card will dynamically reconfigure the loop into two single loops, effectively isolating the failed disk. If the disk is part of a RAID array, the adapter card will automatically regenerate the missing disk using the remaining data and parity disks to the spare disk. Once the failed disk has been replaced, the loop will automatically be reconfigured into full duplex operation, and the replaced disk will become a new spare.

Spatial reuse

Spatial reuse allows domains to be set up on the loop. A *domain* means that one or more groups of disks belong to one of the two adapter cards, as is the case during normal operation. The benefit of this is that each adapter card can talk to its domains (or disk groups) using only part of the loop. The use of domains allows each adapter card to operate at maximum capability because it is not limited by I/O operations from the other adapter. Theoretically, each adapter card could drive its domains at 160 MB/s, giving 320 MB/s throughput on a single loop! The benefit of domains may reduce slightly over time, due to disk failures causing the groups to become intermixed, but the main benefits of spatial reuse will still apply.

If a cluster should fail, the remaining cluster device adapter will own all the domains on the loop, thus allowing full data access to continue.

3.14 RAID-10

- RAID-10 configurations:
  - First RAID-10 rank configured in the loop will be: 3 + 3 + 2S
  - Additional RAID-10 ranks configured in the loop will be 4 + 4
  - For a loop with an intermixed capacity, the ESS will assign two spares for each capacity. This means there will be one 3+3+2S array per capacity

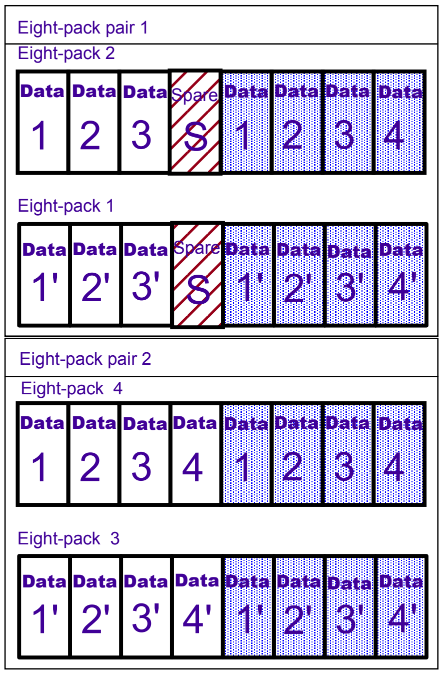


Figure 3-14 RAID-10

RAID-10

RAID-10 is also known as RAID 0+1 because it is a combination of RAID 0 (striping) and RAID 1 (mirroring). The striping optimizes the performance by striping volumes across several disk drives (in the ESS Model 800 implementation, three or four DDMs). RAID 1 is the protection against a disk failure provided by having a mirrored copy of each disk. By combining the two, RAID 10 provides data protection and I/O performance.

Array

A *disk array* is a group of disk drive modules (DDMs) that are arranged in a relationship, for example, a RAID 5 or a RAID 10 array. For the ESS, the arrays are built upon the disks of the disk eight-packs.

Disk eight-pack

The physical storage capacity of the ESS is materialized by means of the disk eight-packs. These are sets of eight DDMs that are installed in pairs in the ESS. Two disk eight-packs provide for two disk groups —four DDMs from each disk eight-pack. These disk groups can be configured as either RAID-5 or RAID-10 ranks.

Spare disks

The ESS requires that a loop have a minimum of two spare disks to enable sparing to occur. The sparing function of the ESS is automatically initiated whenever a DDM failure is detected on a loop and enables regeneration of data from the failed DDM onto a hot spare DDM.

A hot DDM *spare pool* consisting of two drives, created with one 3+3+2S array (RAID 10), is created for each drive size on an SSA loop. Therefore, if only one drive size is installed on a loop, only two spares are required. The hot sparing function is managed at the SSA loop level. SSA will spare to a larger capacity DDM on the loop in the very uncommon situation that no spares are available on the loop for a given capacity.

Figure 3-14 on page 67 shows the following:

1. In eight-pack pair 1, the array consists of three data drives mirrored to three copy drives. The remaining two drives are used as spares.
2. In eight-pack pair 2, the array consists of four data drives mirrored to four copy drives.

3.15 Storage balancing with RAID-10

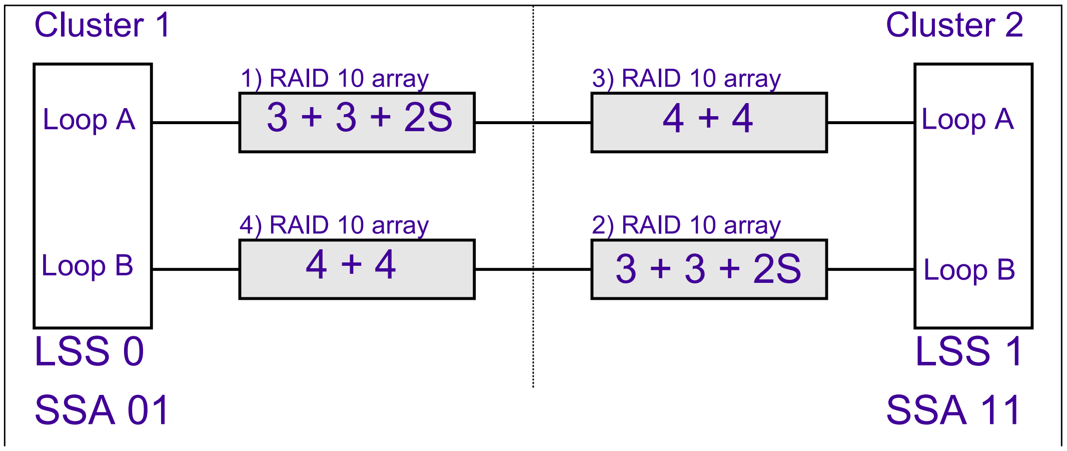


Figure 3-15 Storage balancing with RAID-10

Logical Storage Subsystem (LSS)

The Logical Storage Subsystem (or Logical Subsystem) is a logical structure that is internal to the ESS. It is a logical construct that groups up to 256 logical volumes (logical volumes are defined during the logical configuration procedure) of the same disk format (CKD or FB), and it is identified by the ESS with a unique ID. Although the LSS relates directly to the logical control unit (LCU) concept of the ESCON and FICON architectures, it does not directly relate to SCSI and FCP addressing.

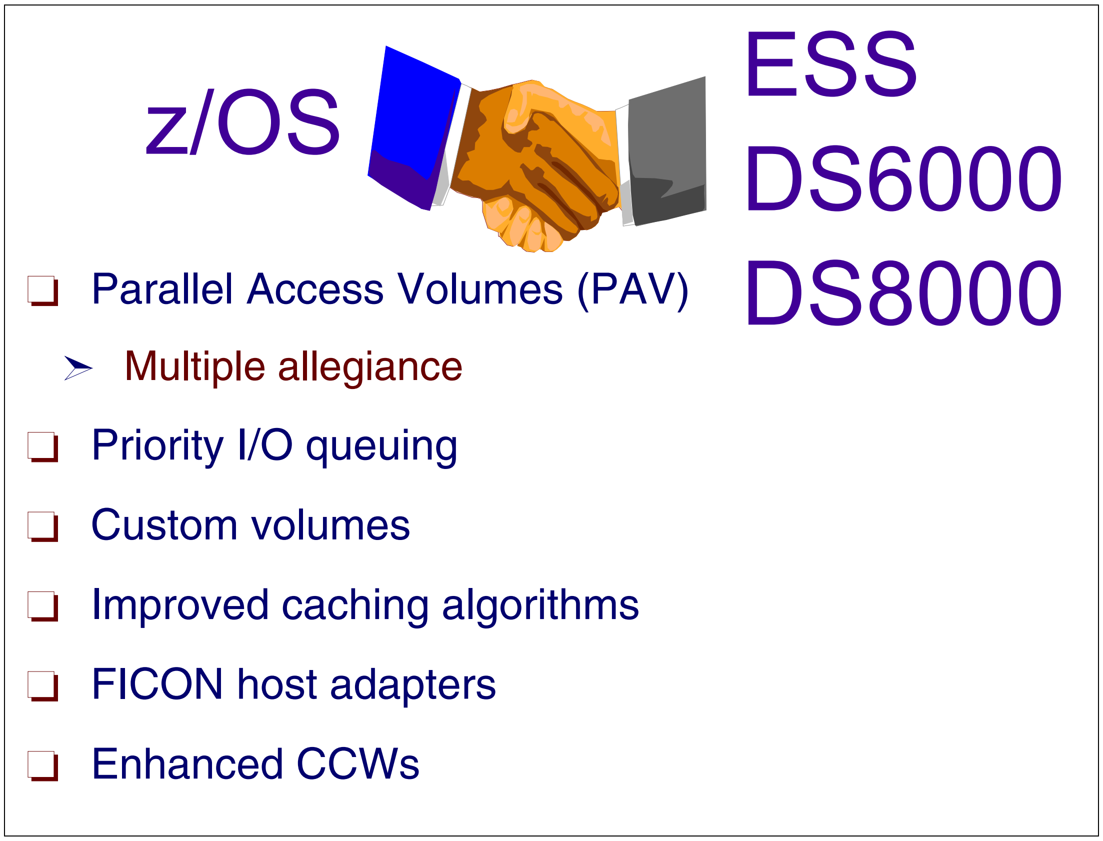
ESS storage balancing

For performance reasons, you should try to allocate storage on the ESS so that it is equally balanced across both clusters and among the SSA loops. One way to accomplish this is to assign two arrays (one from loop A and one from loop B) to each Logical Subsystem. To achieve this you can follow this procedure when configuring RAID-10 ranks:

1. Configure the first array for LSS 0/loop A. This will be a 3 + 3 + 2S array.
2. Configure the first array for LSS 1/loop B. This will also be a 3 + 3 + 2S array.
3. Configure the second array for LSS1/loop A. This will now be a 4 + 4.
4. Configure the second array for LSS 0/loop B. This will also now be a 4 + 4.

Figure 3-15 illustrates the results of this configuration procedure.

3.16 ESS performance features



z/OS

ESS
DS6000
DS8000

- ❑ Parallel Access Volumes (PAV)
  - ➢ Multiple allegiance
- ❑ Priority I/O queuing
- ❑ Custom volumes
- ❑ Improved caching algorithms
- ❑ FICON host adapters
- ❑ Enhanced CCWs

Figure 3-16 ESS performance features

Parallel access volumes (PAV) and multiple allegiance

Traditional S/390 architecture does not allow more than one I/O operation to the same S/390 device because such devices can only handle, physically, one I/O operation at a time. However, in modern DASD subsystems like ESS, DS6000, and DS8000, the device (such as a 3390) is only a logical view. The contents of this logical device are spread in HDA RAID arrays and in caches. Therefore, it is technically possible to have more than one I/O operation towards the same logical device. Parallel access volumes (PAVs) can provide significant performance enhancements in IBM System z environments by enabling simultaneous processing for multiple I/O operations to the same logical volume. Changes are made in z/OS (in IOS code), in the channel subsystem (SAP®), and in ESS, DS6000, and DS8000 in order to allow more than one I/O operation on the same logical device. This is called parallel I/O, and it has two flavors:

- ► Parallel Access Volume (PAV), when the concurrent I/Os originate from the same z/OS image
- ► Multiple allegiance (MA) is the capability to support I/O requests from multiple systems—one per system—to be concurrently active against the same logical volume if they do not conflict with each other. Conflicts occur when two or more I/O requests require access to overlapping extents (an extent is a contiguous range of tracks) on the volume, and at least one of the I/O requests involves writing of data. Requests involving writing of data can execute concurrently with other requests as long as they operate on nonoverlapping extents on the volume. Conflicting requests are internally queued in ESS. Read requests can always execute concurrently regardless of their extents. Without the

MA capability, ESS would generate a busy indication for the volume whenever one of the systems issues a request against the volume. This would cause the I/O requests to be queued within the channel subsystem (CSS).

However, this concurrency can be achieved as long as no data accessed by one channel program can be altered through the actions of another channel program.

To implement PAV, IOS introduces the concept of *alias addresses*. Instead of one UCB per logical volume, an MVS host can now use several UCBs for the same logical volume. Apart from the conventional Base UCB, alias UCBs can be defined and used by z/OS to issue I/Os in parallel to the same logical volume device.

I/O priority queuing

Prior to ESS, IOS kept the UCB I/O pending requests in a queue named IOSQ. The priority order of the I/O request in this queue—when the z/OS image is in goal mode—is controlled by WLM, depending on the transaction owning the I/O request. There was no concept of priority queuing within the internal queues of the I/O control units; instead, the queue regime was FIFO.

With ESS, it is possible to have this queue concept internally; I/O Priority Queueing in ESS has the following properties:

- I/O can be queued with the ESS in priority order.
- WLM sets the I/O priority when running in goal mode.
- There is I/O priority for systems in a sysplex.
- Each system gets a fair share.

Custom volumes

Custom volumes provides the possibility of defining small size 3390 or 3380 volumes. This causes less contention on a volume. Custom volumes is designed for high activity data sets. Careful size planning is required.

Improved caching algorithms

With its effective caching algorithms, IBM TotalStorage Enterprise Storage Server Model 800 is able to minimize wasted cache space and reduce disk drive utilization, thereby reducing its back-end traffic. The ESS Model 800 has a maximum cache size of 64 GB, and the NVS standard size is 2 GB.

The ESS manages its cache in 4 KB segments, so for small data blocks (4 KB and 8 KB are common database block sizes), minimum cache is wasted. In contrast, large cache segments could exhaust cache capacity while filling up with small random reads. Thus the ESS, having smaller cache segments, is able to avoid wasting cache space for situations of small record sizes that are common in interactive applications.

This efficient cache management, together with the ESS Model 800 powerful back-end implementation that integrates new (optional) 15,000 rpm drives, enhanced SSA device adapters, and twice the bandwidth (as compared to previous models) to access the larger NVS (2 GB) and the larger cache option (64 GB), all integrate to give greater throughput while sustaining cache speed response times.

FICON host adapters

FICON extends the IBM TotalStorage Enterprise Storage Server Model 800's ability to deliver bandwidth potential to the volumes needing it, when they need it.

Performance enhanced channel command words (CCWs)

For the z/OS environments, the ESS supports channel command words (CCWs) that reduce the characteristic overhead associated to the previous (3990) CCW chains. Basically, with these CCWs, the ESS can read or write more data with fewer CCWs. CCW chains using the old CCWs are converted to the new CCWs whenever possible. The cooperation of z/OS software and the ESS provides the best benefits for the application's performance. In other words, in ESS there is less overhead associated with CCW chains by combining tasks into fewer CCWs, introducing Read Track Data and Write Track Data CCWs. They allow reading and writing more data with fewer CCWs. It will be used by z/OS to reduce ESCON protocol for multiple record transfer chains. Measurements on 4 KB records using an EXCP channel program showed a 15 percent reduction in channel overhead for the Read Track Data CCW.

3.17 WLM controlling PAVs

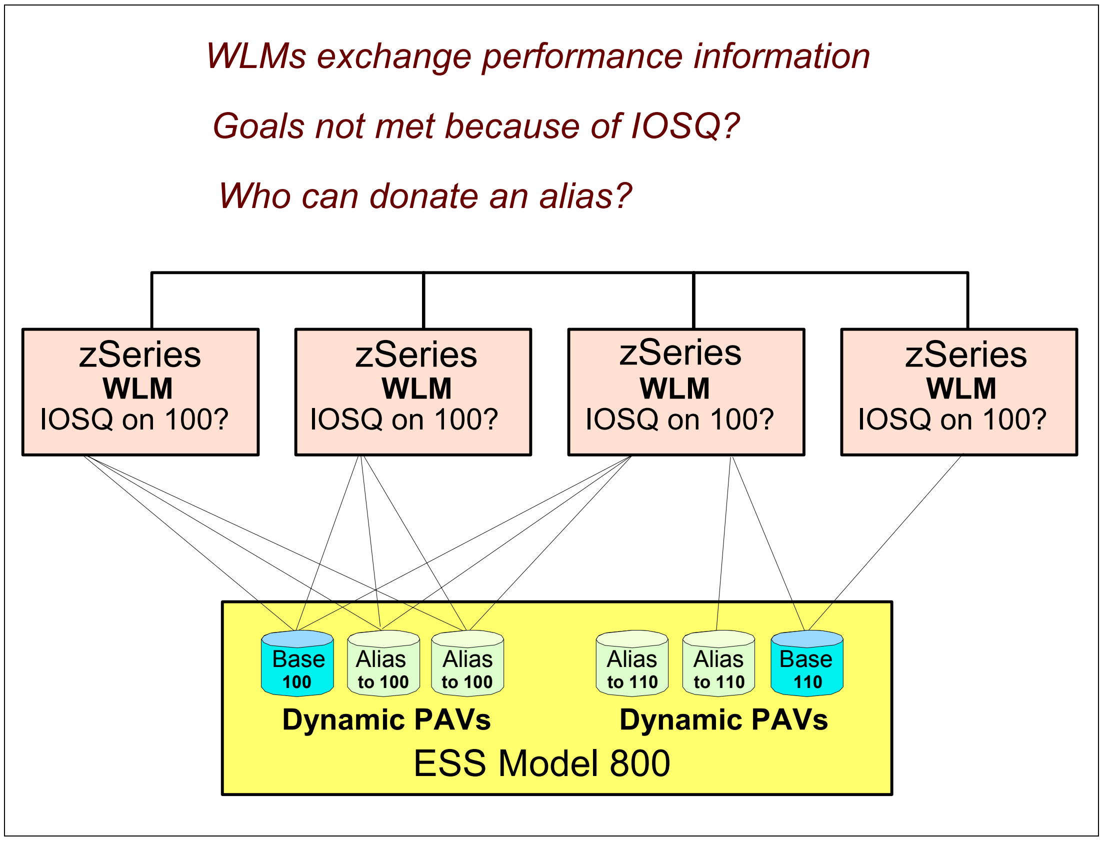


Figure 3-17 WLM controlling PAVs

Workload Manager (WLM)

In the zSeries Parallel Sysplex environments, the z/OS Workload Manager (WLM) controls where work is run and optimizes the throughput and performance of the total system. The ESS provides the WLM with more sophisticated ways to control the I/O across the sysplex. These functions include parallel access to both single-system and shared volumes, and the ability to prioritize the I/O based upon WLM goals. The combination of these features significantly improves performance in a wide variety of workload environments.

Parallel Access Volume (PAV)

Parallel Access Volume is one of the original features that the IBM TotalStorage Enterprise Storage Server brings specifically for z/OS operating systems, helping the zSeries running applications to concurrently share the same logical volumes.

The ability to do multiple I/O requests to the same volume nearly eliminates IOS queue time (IOSQ), one of the major components in z/OS response time. Traditionally, access to highly active volumes has involved manual tuning, splitting data across multiple volumes, and more. With PAV and the Workload Manager, you can almost forget about manual performance tuning. WLM manages PAVs across all members of a sysplex, too. The ESS, in conjunction with z/OS, has the ability to meet the performance requirements on its own.

Alias assignment

It will not always be easy to predict which volumes should have an alias address assigned, and how many. Your software can automatically manage the aliases according to your goals.

z/OS can exploit automatic PAV tuning if you are using WLM in goal mode. z/OS recognizes the aliases that are initially assigned to a base during the Nucleus Initialization Program (NIP) phase. WLM can dynamically tune the assignment of alias addresses. WLM monitors the device performance and is able to dynamically reassign alias addresses from one base to another if predefined goals for a workload are not met. WLM instructs IOS to reassign an alias.

WLM goal mode management in a sysplex

WLM keeps track of the devices utilized by the different workloads, accumulates this information over time, and broadcasts it to the other systems in the same sysplex. If WLM determines that any workload is not meeting its goal due to IOSQ time, WLM attempts to find an alias device that can be reallocated to help this workload achieve its goal.

Through WLM, there are two mechanisms to tune the alias assignment:

- The first mechanism is goal based. This logic attempts to give additional aliases to a PAV device that is experiencing IOS queue delays and is impacting a service class period that is missing its goal. To give additional aliases to the receiver device, a donor device must be found with a less important service class period. A bitmap is maintained with each PAV device that indicates the service classes using the device.
- The second mechanism is to move aliases to high-contention PAV devices from low-contention PAV devices. High-contention devices will be identified by having a significant amount of IOSQ. This tuning is based on efficiency rather than directly helping a workload to meet its goal.

3.18 Parallel Access Volumes (PAVs)

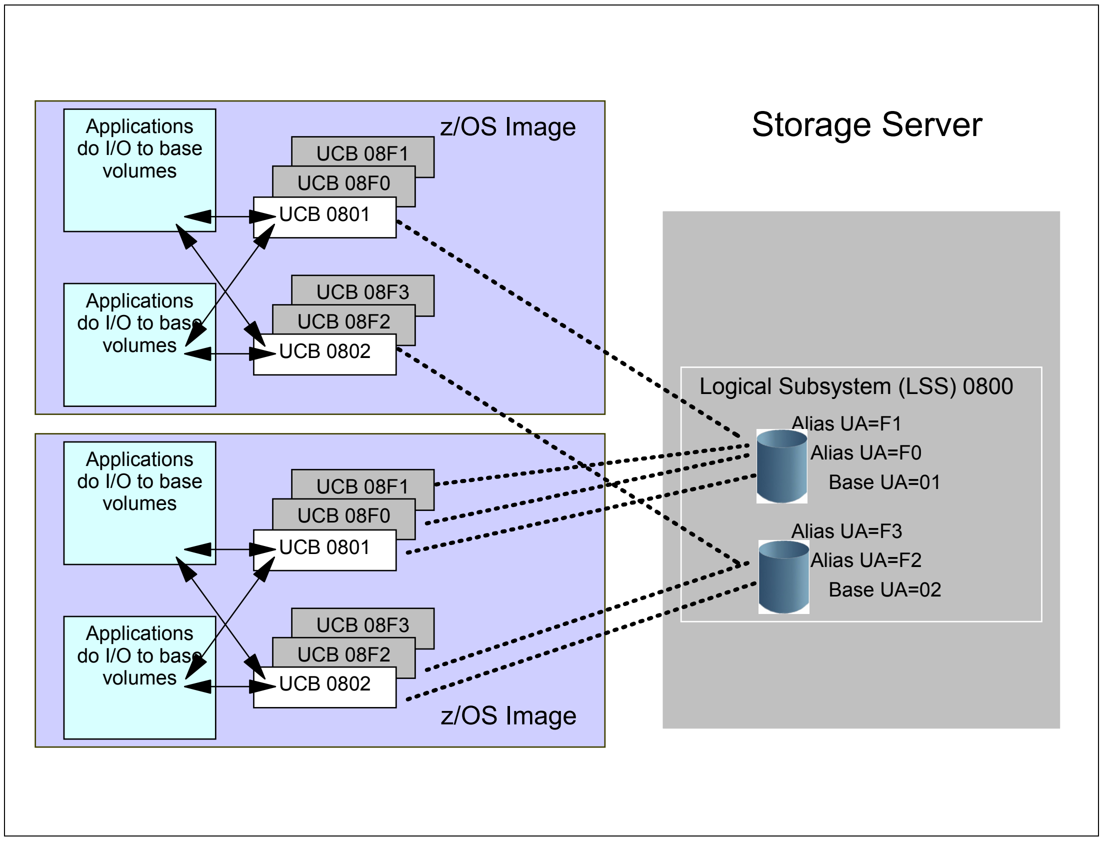


Figure 3-18 Parallel access volumes (PAVs)

Parallel access volume (PAV)

z/OS system IOS maps a device in a unit control block (UCB). Traditionally this I/O device does not support concurrency, being treated as a single resource, serially used. High I/O activity towards the same device can adversely affect performance. This contention is worst for large volumes with many small data sets. The symptom displayed is extended IOSQ time, where the I/O request is queued in the UCB. z/OS cannot attempt to start more than one I/O operation at a time to the device.

The ESS and DS8000 support concurrent data transfer operations to or from the same 3390/3380 devices from the same system. A device (volume) accessed in this way is called a parallel access volume (PAV).

PAV exploitation requires both software enablement and an optional feature on your controller. PAV support must be installed on each controller. It enables the issuing of multiple channel programs to a volume from a single system, and allows simultaneous access to the logical volume by multiple users or jobs. Reads, as well as writes to different extents, can be satisfied simultaneously. The domain of an I/O consists of the specified extents to which the I/O operation applies, which corresponds to the extents of the same data set. Writes to the same domain still have to be serialized to maintain data integrity, which is also the case for reads and write.

The implementation of *N* parallel I/Os to the same 3390/3380 device consumes *N* addresses in the logical controller, thus decreasing the number of possible real devices. Also, UCBs are not prepared to allow multiples I/Os due to software product compatibility issues. Support is then implemented by defining multiple UCBs for the same device.

The UCBs are of two types:

- Base address: This is the actual unit address. There is only one for any volume.
- Alias address: Alias addresses are mapped back to a base device address. I/O scheduled for an alias is executed against the base by the controller. No physical disk space is associated with an alias address. Alias UCBs are stored above the 16 MB line.

PAV benefits

Workloads that are most likely to benefit from PAV functionality being available include:

- Volumes with many concurrently open data sets, such as volumes in a work pool
- Volumes that have a high read to write ratio per extent
- Volumes reporting high IOSQ times

Candidate data sets types:

- Have high read to write ratio
- Have many extents on one volume
- Are concurrently shared by many readers
- Are accessed using media manager or VSAM-extended format (32-byte suffix)

PAVs can be assigned to base UCBs either:

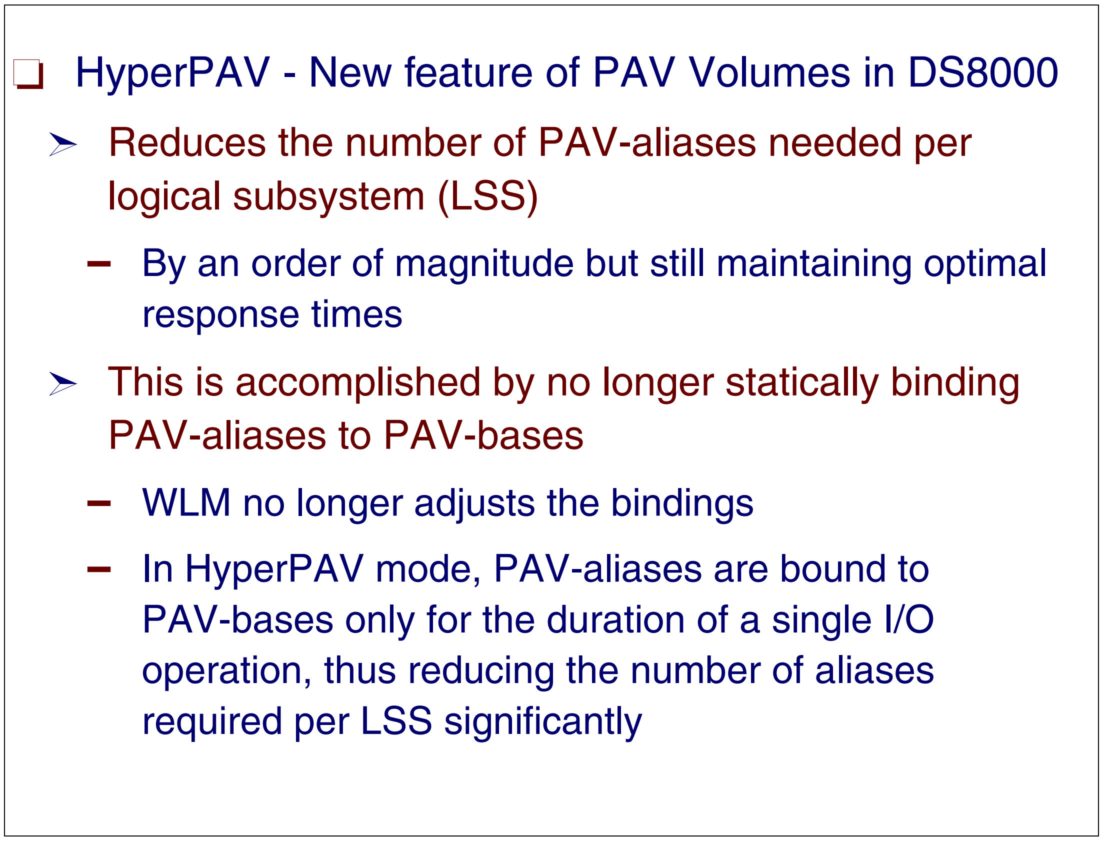
- Manually (static) by the installation.
- Dynamically, WLM can move aliases' UCBs from one base UCB to another base UCB in order to:
  - Balance device utilizations
  - Honor the goal of transactions suffering I/O delays because long IOSQ time. All WLMs in a sysplex must agree with the movement of aliases' UCBs.

However, the dynamic PAV still has some problems:

- Any change must be decided by all WLMs in a sysplex using XCF communication.
- The number of aliases for one device must be equal in all z/OS systems.
- Any change implies a dynamic I/O configuration.

To solve such problems HyperPAV was introduced. With HyperPAV all aliases' UCBs are located in a pool and are used dynamically by IOS.

3.19 HyperPAV feature for DS8000 series



- HyperPAV - New feature of PAV Volumes in DS8000
  - Reduces the number of PAV-aliases needed per logical subsystem (LSS)
    - By an order of magnitude but still maintaining optimal response times
  - This is accomplished by no longer statically binding PAV-aliases to PAV-bases
    - WLM no longer adjusts the bindings
    - In HyperPAV mode, PAV-aliases are bound to PAV-bases only for the duration of a single I/O operation, thus reducing the number of aliases required per LSS significantly

Figure 3-19 HyperPAV implementation

DS8000 feature

HyperPAV is an optional feature on the DS8000 series, available with the HyperPAV indicator feature number 0782 and corresponding DS8000 series function authorization (2244-PAV HyperPAV feature number 7899). HyperPAV also requires the purchase of one or more PAV licensed features and the FICON/ESCON Attachment licensed feature. The FICON/ESCON Attachment licensed feature applies only to the DS8000 Turbo Models 931, 932, and 9B2. HyperPAV allows many DS8000 series users to benefit from enhancements to PAV with support for HyperPAV.

HyperPAV allows an alias address to be used to access any base on the same control unit image per I/O base. This capability also allows different HyperPAV hosts to use one alias to access different bases, which reduces the number of alias addresses required to support a set of bases in a System z environment with no latency in targeting an alias to a base. This functionality is also designed to enable applications to achieve equal or better performance than is possible with the original PAV feature alone, while also using the same or fewer z/OS resources. The HyperPAV capability is offered on z/OS V1R6 and later.

3.20 HyperPAV implementation

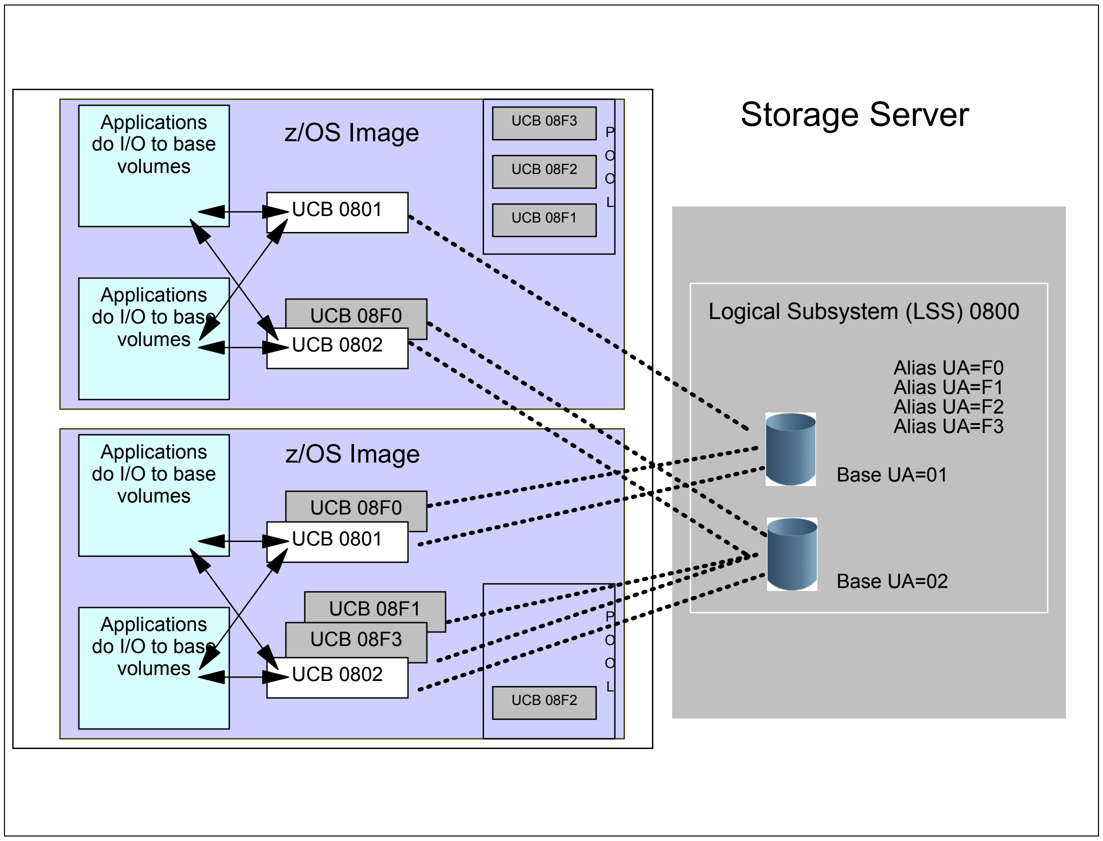


Figure 3-20 HyperPAV implementation using a pool of aliases

HyperPAV feature

With the IBM System Storage™ DS8000 Turbo model and the IBM server synergy feature, the HyperPAV together with PAV, Multiple Allegiance, and support for IBM System z MIDAW facility can dramatically improve performance and efficiency for System z environments.

With HyperPAV technology:

- z/OS uses a pool of UCB aliases.
- As each application I/O is requested, if the base volume is busy with another I/O:
  - z/OS selects a free alias from the pool, quickly binds the alias device to the base device, and starts the I/O.
  - When the I/O completes, the alias device is used for another I/O on the LSS or is returned to the free alias pool.

If too many I/Os are started simultaneously:

- z/OS will queue the I/Os at the LSS level.
- When an exposure frees up that can be used for queued I/Os, they are started.
- Queued I/O is done within assigned I/O priority.

For each z/OS image within the sysplex, aliases are used independently. WLM is not involved in alias movement so it does not need to collect information to manage HyperPAV aliases.

Note: HyperPAV was introduced and integrated in z/OS V1R9 and is available in z/OS V1R8 with APAR OA12865.

3.21 ESS copy services

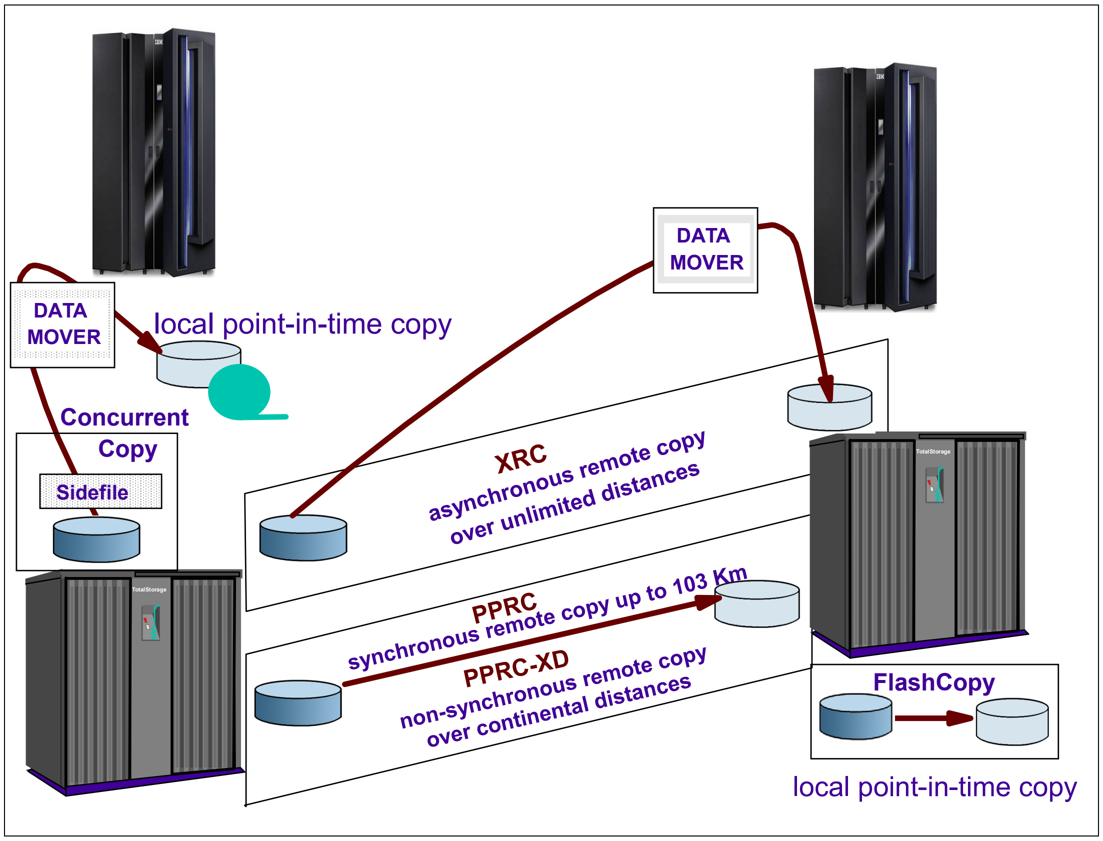

DATA MOVER
local point-in-time copy
Concurrent Copy
Sidefile
DATA MOVER
XRC
asynchronous remote copy over unlimited distances
PPRC
synchronous remote copy up to 103 Km
PPRC-XD
non-synchronous remote copy over continental distances
FlashCopy
local point-in-time copy

Figure 3-21 ESS copy services

DFSMS copy services

DFSMS provides Advanced Copy Services that include a hardware and software solution to help you manage and protect your data. These solutions help ensure that your data remains available 24 hours a day, seven days a week. Advanced Copy Services provide solutions to disaster recovery, data migration, and data duplication. Many of these functions run on the IBM TotalStorage Enterprise Storage Server (ESS). With DFSMS, you can perform the following data management functions:

- Use remote copy to prepare for disaster recovery
- Move your PPRC data more easily

Remote copy provides two options that enable you to maintain a current copy of your data at a remote site. These two options are used for disaster recovery and workload migration:

- Extended remote copy (XRC)
- Peer-to-peer remote copy (PPRC)

There are two types of copy:

- An instantaneous copy where all the late updates in the primary are not copied. It is used for fast backups and data replication in general. The examples in ESS are Concurrent Copy and Flash Copy.
- Mirroring, a never-ending copy where all the updates are mirrored as fast as possible. It is used for disaster recovery and planned outages. The example in ESS are Enhanced PPRC service and XRC.

Peer-to-peer remote copy (PPRC)

PPRC is a hardware solution which provides rapid and accurate disaster recovery as well as a solution to workload movement and device migration. Updates made on the primary DASD volumes are synchronously shadowed to the secondary DASD volumes. The local storage subsystem and the remote storage subsystem are connected through a communications link called a PPRC path. You can use one of the following protocols to copy data using PPRC:

- ESCON
- Fibre Channel Protocol

> **Note:** Fibre Channel Protocol is supported only on ESS Model 800 with the appropriate licensed internal code (LIC) level and the PPRC Version 2 feature enabled.

PPRC provides a synchronous volume copy across ESS controllers. The copy is done from one controller (the one having the primary logical device) to the other (having the secondary logical device). It is synchronous because the task doing the I/O receives the CPU back with the guarantee that the copy was executed. There is a performance penalty for distances longer than 10 km. PPRC is used for disaster recovery, device migration, and workload migration; for example, it enables you to switch to a recovery system in the event of a disaster in an application system.

You can issue the `CQUERY` command to query the status of one volume of a PPRC volume pair or to collect information about a volume in the simplex state. The `CQUERY` command is modified and enabled to report on the status of S/390-attached CKD devices.

See *z/OS DFSMS Advanced Copy Services,* SC35-0428*,* for further information about the PPRC service and the `CQUERY` command.

Peer-to-peer remote copy extended distance (PPRC-XD)

When you enable the PPRC extended distance feature (PPRC-XD), the primary and recovery storage control sites can be separated by long distances. Updates made to a PPRC primary volume are sent to a secondary volume asynchronously, thus requiring less bandwidth.

If you are trying to decide whether to use synchronous or asynchronous PPRC, consider the differences between the two modes:

- When you use synchronous PPRC, no data loss occurs between the last update at the primary system and the recovery site, but it increases the impact to applications and uses more resources for copying data.
- Asynchronous PPRC using the extended distance feature reduces impact to applications that write to primary volumes and uses less resources for copying data, but data might be lost if a disaster occurs. To use PPRC-XD as a disaster recovery solution, customers need to periodically synchronize the recovery volumes with the primary site and make backups to other DASD volumes or tapes.

PPRC Extended Distance (PPRC-XD) is a non-synchronous version of PPRC. This means that host updates to the source volume are not delayed by waiting for the update to be confirmed in the secondary volume. It also means that the sequence of updates on the secondary volume is not guaranteed to be the same as on the primary volume.

PPRC-XD is an excellent solution for:

- Remote data copy
- Remote data migration
- Offsite backup
- Transmission of inactive database logs

- Application disaster recovery solutions based on periodic point-in-time (PiT) copies of the data, if the application tolerates short interruptions (application quiesce)

PPRC-XD can operate at very long distances (such as continental distances), well beyond the 103 km supported for PPRC synchronous transmissions—and with minimal impact on the application. The distance is limited only by the network and channel extender technology capabilities.

Extended remote copy (XRC)

XRC combines hardware and software to provide continuous data availability in a disaster recovery or workload movement environment. XRC provides an asynchronous remote copy solution for both system-managed and non-system-managed data to a second, remote location.

XRC relies on the IBM TotalStorage Enterprise Storage Server, IBM 3990, RAMAC Storage Subsystems, and DFSMSdfp. The 9393 RAMAC Virtual Array (RVA) does not support XRC for source volume capability.

XRC relies on the system data mover, which is part of DFSMSdfp. The system data mover is a high-speed data movement program that efficiently and reliably moves large amounts of data between storage devices. XRC is a continuous copy operation, and it is capable of operating over long distances (with channel extenders). It runs unattended, without involvement from the application users. If an unrecoverable error occurs at your primary site, the only data that is lost is data that is in transit between the time when the primary system fails and the recovery at the recovery site.

You can implement XRC with one or two systems. Let us suppose that you have two systems: an application system at one location, and a recovery system at another. With these two systems in place, XRC can automatically update your data on the remote disk storage subsystem as you make changes to it on your application system. You can use the XRC suspend/resume service for planned outages. You can still use this standard XRC service on systems attached to the ESS if these systems are installed with the toleration or transparency support.

Coupled Extended Remote Copy (CXRC) allows XRC sessions to be coupled together to guarantee that all volumes are consistent across all coupled XRC sessions. CXRC can manage thousands of volumes. IBM TotalStorage XRC Performance Monitor provides the ability to monitor and evaluate the performance of a running XRC configuration.

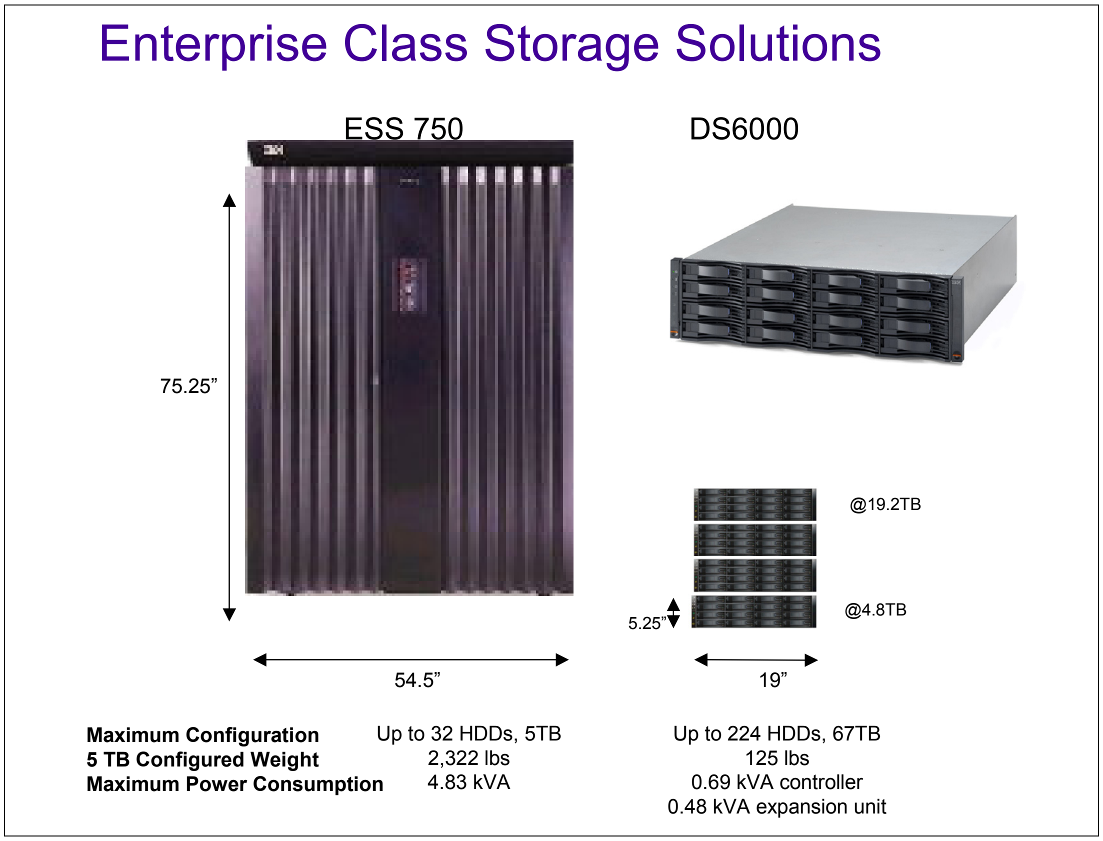
Concurrent copy

Concurrent copy is an extended function that enables data center operations staff to generate a copy or a dump of data while applications are updating that data. Concurrent copy delivers a copy of the data, in a consistent form, as it existed before the updates took place.

FlashCopy service

FlashCopy is a point-in-time copy services function that can quickly copy data from a source location to a target location. FlashCopy enables you to make copies of a set of tracks, with the copies immediately available for read or write access. This set of tracks can consist of an entire volume, a data set, or just a selected set of tracks. The primary objective of FlashCopy is to create a copy of a source volume on the target volume. This copy is called a *point-in-time copy*. Access to the point-in-time copy of the data on the source volume is through reading the data from the target volume. The actual point-in-time data that is read from the target volume might or might not be physically stored on the target volume. The ESS FlashCopy service is compatible with the existing service provided by DFSMSdss. Therefore, you can invoke the FlashCopy service on the ESS with DFSMSdss.

3.22 IBM TotalStorage DS6000


Enterprise Class Storage Solutions

	ESS 750	DS6000
Maximum Configuration	Up to 32 HDDs, 5TB	Up to 224 HDDs, 67TB
5 TB Configured Weight	2,322 lbs	125 lbs
Maximum Power Consumption	4.83 kVA	0.69 kVA controller 0.48 kVA expansion unit

Figure 3-22 IBM TotalStorage DS6000

IBM TotalStorage DS6000

The IBM TotalStorage DS6000 series is designed to deliver the resiliency, performance, and many of the key features of the IBM TotalStorage Enterprise Storage Server (ESS) in a small, modular package.

The DS6000 series offers high scalability while maintaining excellent performance. With the DS6800 (Model 1750-511), you can install up to 16 disk drive modules (DDMs). The minimum storage capability with 8 DDMs is 584 GB. The maximum storage capability with 16 DDMs for the DS6800 model is 4.8 TB. If you want to connect more than 16 disks, you can use up to 13 DS6000 expansion units (Model 1750-EX1) that allow a maximum of 224 DDMs per storage system and provide a maximum storage capability of 67 TB.

DS6000 specifications

Table 3-3 summarizes the DS6000 features.

Table 3-3 DS6000 specifications

	DS6000
Controllers	Dual active
Max cache	4 GB
Max Host Ports	8-Ports; 2Gb FC/FICON

	DS6000
Max Hosts	1024
Max Storage / Disks	224
Disk Types	FC 10K: 146 GB, 300 GB FC 15K: 73 GB
Max Expansion Mod	13
Max Disk Loops	4 (2 dual redundant)
Max LUNs	8192(up to 2 TB LUN size)
RAID Levels	5, 10
RAID Array Sizes	4 or 8 drives
Operating Systems	z/OS, i5/OS®, OS/400, AIX, SUN Solaris™, HP UX, VMWare, Microsoft® Windows, Linux
Packaging	3U – Controller & Expansion Drawers
Power consumption	Controller: 0.69 kVA Expansion drawer: 0.48 kVA

Modular scalability

The DS6000 is modularly scalable, with optional expansion enclosure, to add capacity to help meet your growing business needs. The scalability comprises:

- Flexible design to accommodate on demand business environments
- Ability to make dynamic configuration changes
  - Add disk drives in increments of 4
  - Add storage expansion units
- Scale capacity to over 67 TB

DS6800 (Model 1750-511)

The DS6800 is a self-contained 3U enclosure that can be mounted in a standard 19-inch rack. The DS6800 comes with authorization for up to 16 internal FC DDMs, offering up to 4.8 TB of storage capability. The DS6800 allows up to 13 DS6000 expansion enclosures to be attached. A storage system supports up to 224 disk drives for a total of up to 67.2 TB of storage.

The DS6800 offers the following features:

- Two FC controller cards.
- PowerPC® 750GX 1 GHz processor.
- 4 GB of cache.
- Two battery backup units (one per each controller card).
- Two AC/DC power supplies with imbedded enclosure cooling units.
- Eight 2 Gbps device ports.
- Connectivity with the availability of two to eight fibre channel/FICON host ports. The host ports auto-negotiate to either 2 Gbps or 1 Gbps link speeds.
- Attachment to 13 DS6000 expansion enclosures.

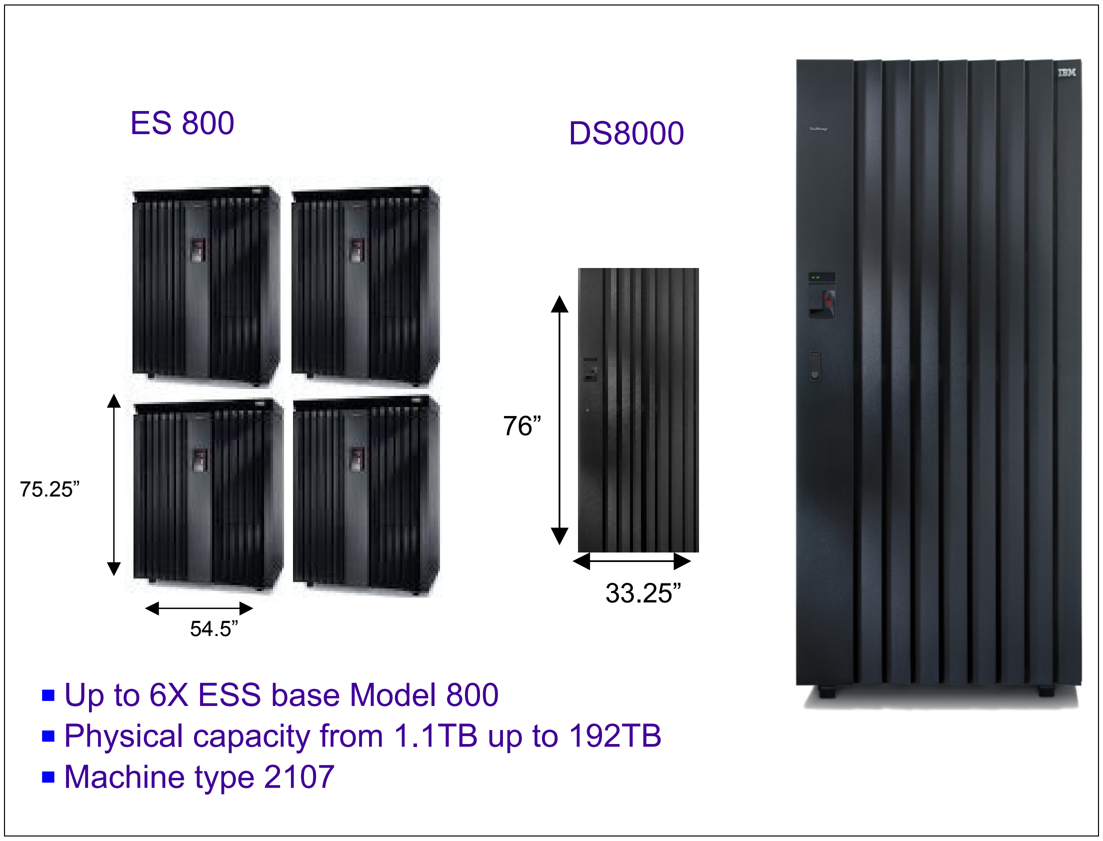
DS6000 expansion enclosure (Model 1750-EX1)

The 3U DS6000 expansion enclosure can be mounted in a standard 19-inch rack. The front of the enclosure contains the docking sites where you can install up to 16 DDMs.

The DS6000 expansion enclosure contains the following features:

- Two expansion controller cards. Each controller card provides the following:
  - Two 2 Gbps inbound ports
  - Two 2 Gbps outbound ports
  - One FC switch per controller card
- Controller disk enclosure that holds up to 16 FC DDMs
- Two AC/DC power supplies with imbedded enclosure cooling units
- Supports attachment to DS6800

3.23 IBM TotalStorage DS8000



- Up to 6X ESS base Model 800
- Physical capacity from 1.1TB up to 192TB
- Machine type 2107

Figure 3-23 IBM TotalStorage DS8000

IBM TotalStorage DS8000

IBM TotalStorage DS8000 is a high-performance, high-capacity series of disk storage that is designed to support continuous operations. DS8000 series models (machine type 2107) use the *IBM POWER5™* server technology that is integrated with the *IBM Virtualization Engine™* technology. DS8000 series models consist of a storage unit and one or two management consoles, two being the recommended configuration. The graphical user interface (GUI) or the command-line interface (CLI) allows you to *logically partition storage* (create storage lpars) and use the built-in Copy Services functions. For high availability, hardware components are redundant.

The current physical storage capacity of the DS8000 series system can range from 1.1 TB to 192 TB of physical capacity, and it has an architecture designed to scale to over 96 petabytes.

DS8000 models

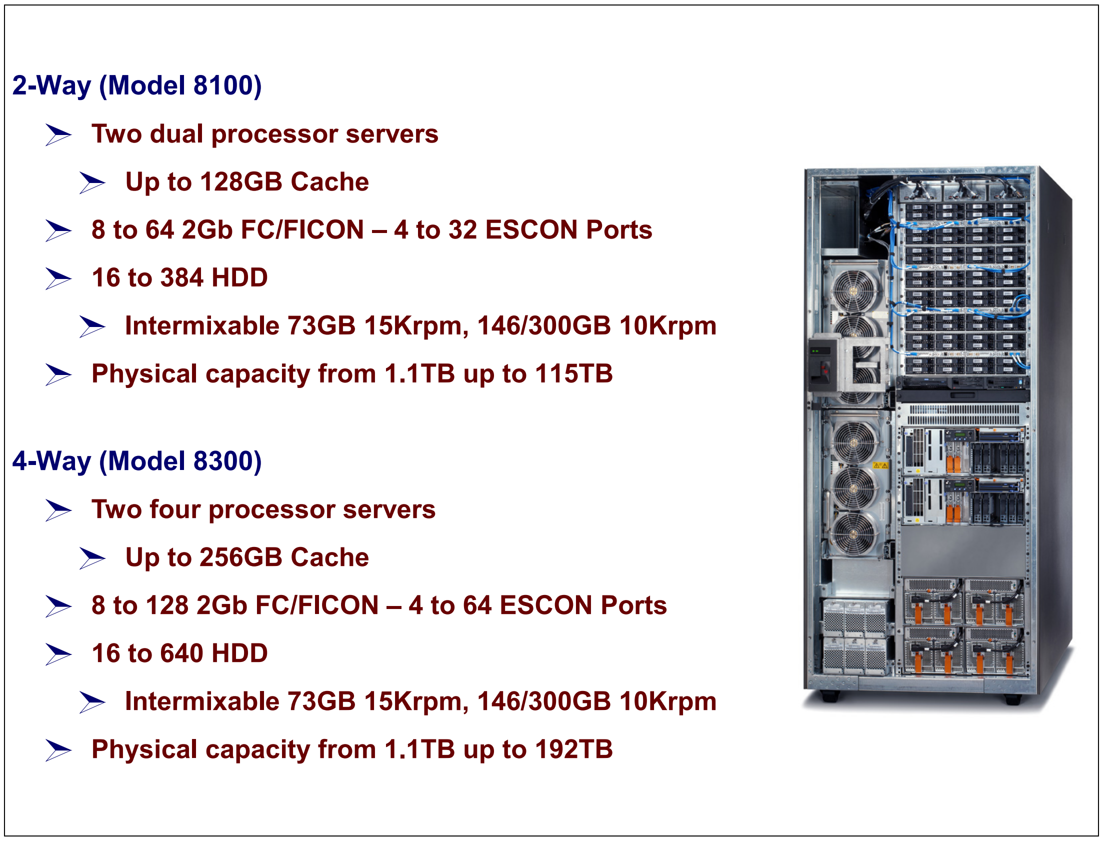
The DS8000 series offers various choices of base and expansion models, so you can configure storage units that meet your performance and configuration needs.

- DS8100
  The DS8100 (Model 921) features a dual *two-way processor* complex and support for one expansion frame.
- DS8300
  The DS8300 (Models 922 and 9A2) features a dual four-way processor complex and

support for one or two expansion frames. The Model 9A2 supports two IBM TotalStorage System LPARs (Logical Partitions) in one storage unit.

The DS8000 expansion frames (Models 92E and 9AE) expand the capabilities of the base models. You can attach the Model 92E to either the Model 921 or the Model 922 to expand their capabilities. You can attach the Model 9AE to expand the Model 9A2.

3.24 DS8000 hardware overview



2-Way (Model 8100)

- Two dual processor servers
  - Up to 128GB Cache
- 8 to 64 2Gb FC/FICON – 4 to 32 ESCON Ports
- 16 to 384 HDD
  - Intermixable 73GB 15Krpm, 146/300GB 10Krpm
- Physical capacity from 1.1TB up to 115TB

4-Way (Model 8300)

- Two four processor servers
  - Up to 256GB Cache
- 8 to 128 2Gb FC/FICON – 4 to 64 ESCON Ports
- 16 to 640 HDD
  - Intermixable 73GB 15Krpm, 146/300GB 10Krpm
- Physical capacity from 1.1TB up to 192TB

Figure 3-24 DS8000 models

DS8100 (Model 921)

The IBM TotalStorage DS8100, which is Model 921, offers features that include the following:

- Dual two-way processor complex
- Up to 128 disk drives, for a maximum capacity of 38.4 TB
- Up to 128 GB of processor memory (cache)
- Up to 16 fibre-channel/FICON or ESCON host adapters

The DS8100 model can support *one* expansion frame. With one expansion frame, you can expand the capacity of the Model 921 as follows:

- Up to 384 disk drives, for a maximum capacity of 115.2 TB

DS8300 (Models 922 and 9A2)

IBM TotalStorage DS8300 models (Model 922 and Model 9A2) offer higher performance and capacity than the DS8100. The Model 9A2 also enables you to create two storage system LPARs (or images) within the same storage unit.

Both DS8300 models offer the following features:

- Dual four-way processor complex
- Up to 128 disk drives, for a maximum capacity of 38.4 TB
- Up to 256 GB of processor memory (cache)

- Up to 16 fibre-channel/FICON or ESCON host adapters

The DS8300 models can support either one or two expansion frames. With expansion frames, you can expand the Model 922 and 9A2 as follows:

- With *one* expansion frame, you can support the following expanded capacity and number of adapters:
  - Up to 384 disk drives, for a maximum capacity of 115.2 TB
  - Up to 32 fibre-channel/FICON or ESCON host adapters
- With *two* expansion frames, you can support the following expanded capacity:
  - Up to 640 disk drives, for a maximum capacity of 192 TB

3.25 Storage systems LPARs

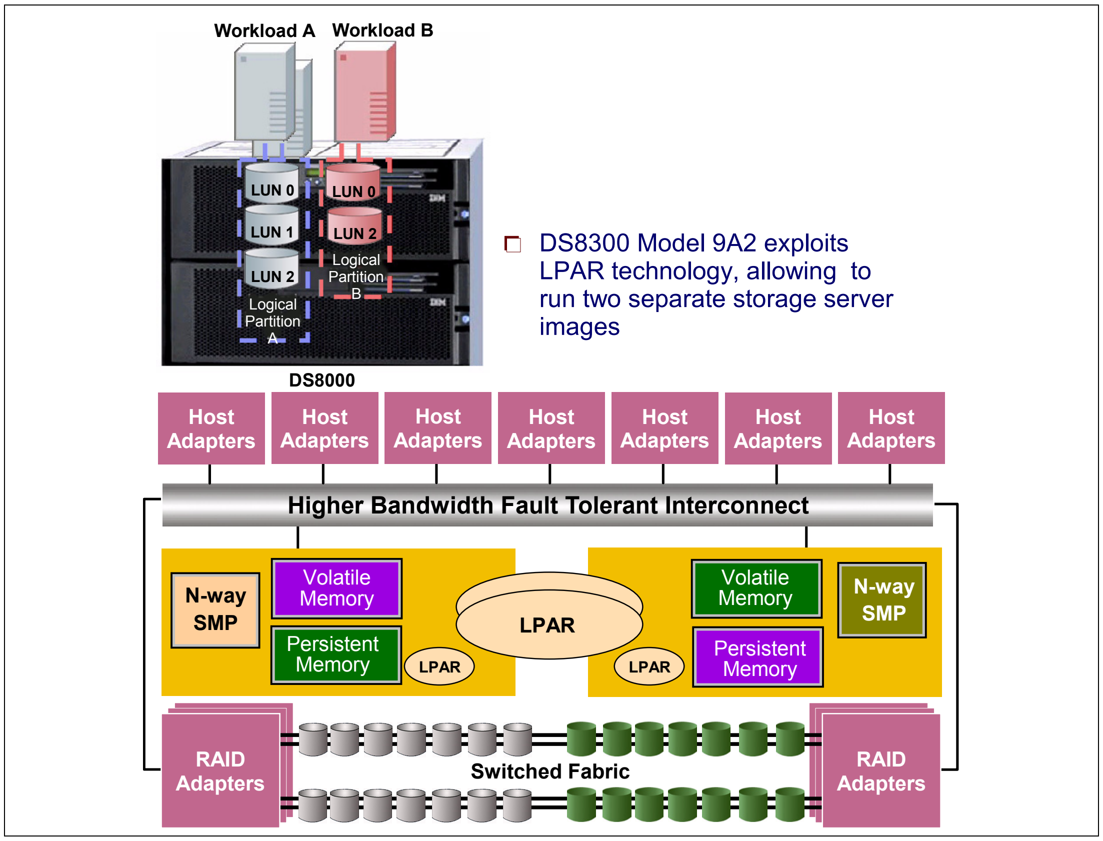


Figure 3-25 Storage systems LPARs

LPAR overview

A logical partition (LPAR) is a subset of logical resources that is capable of supporting an operating system. It consists of CPUs, memory, and I/O slots that are a subset of the pool of available resources within a system. These resources are assigned to the logical partition. Isolation between LPARs is provided to prevent unauthorized access between partition boundaries.

Storage systems LPARs

The DS8300 Model 9A2 exploits LPAR technology, allowing you to run two separate storage server images.

Each Storage System LPAR has access to:

- 50 percent of the processors
- 50 percent of the processor memory
- Up to 16 host adapters
- Up to 320 disk drives (up to 96 TB of capacity)

With these separate resources, each Storage System LPAR can run the same or different versions of microcode, and can be used for completely separate production, test, or other unique storage environments within this single physical system. This may enable storage consolidations where separate storage subsystems were previously required, helping to increase management efficiency and cost effectiveness.

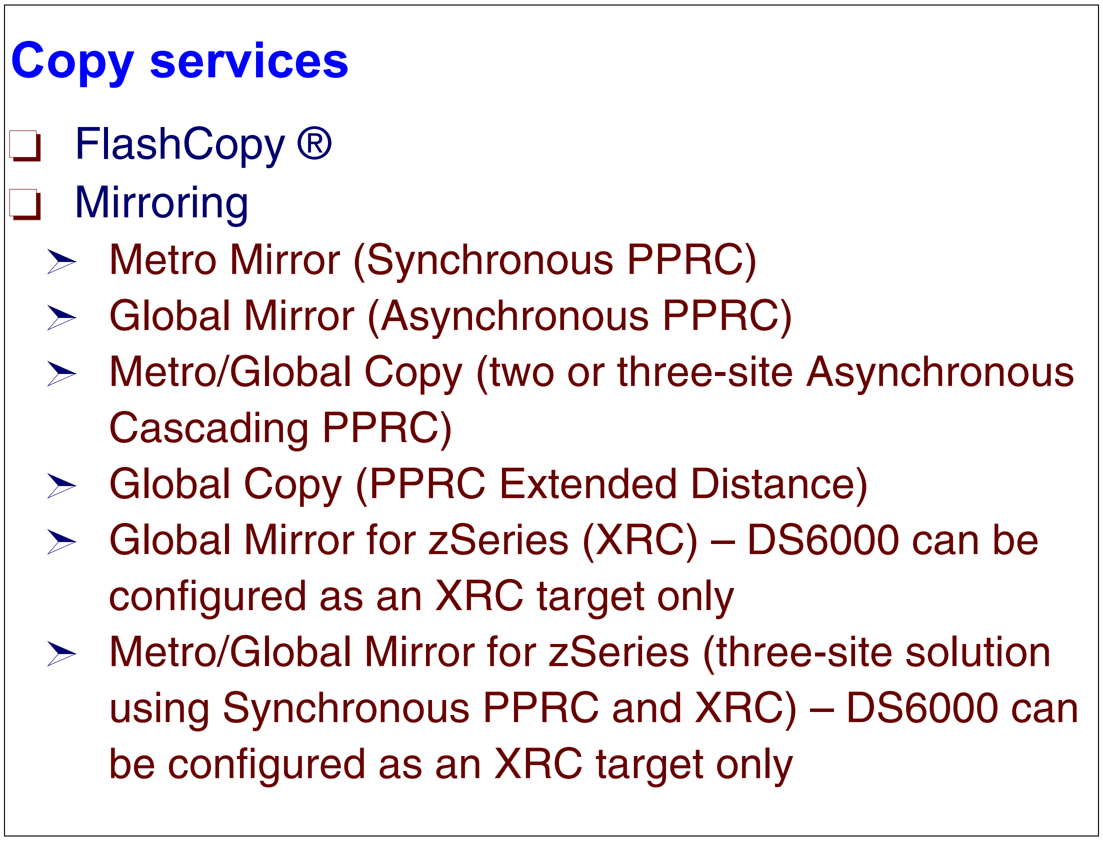
DS8000 addressing capability

Table 3-3 shows the DS8000 addressing capability in comparison to an ESS 800.

Table 3-4 Addressing capability comparison

	ESS 800	DS8000	DS8000 w/LPAR
Max Logical Subsystems	32	255	510
Max Logical Devices	8K	64K	128K
Max Logical CKD Devices	4K	64K	128K
Max Logical FB Devices	4K	64K	128K
Max N-Port Logins/Port	128	509	509
Max N-Port Logins	512	8K	16K
Max Logical Paths/FC Port	256	2K	2K
Max Logical Paths/CU Image	256	512	512
Max Path Groups/CU Image	128	256	256

3.26 IBM TotalStorage Resiliency Family



Copy services

- FlashCopy ®
- Mirroring
  - Metro Mirror (Synchronous PPRC)
  - Global Mirror (Asynchronous PPRC)
  - Metro/Global Copy (two or three-site Asynchronous Cascading PPRC)
  - Global Copy (PPRC Extended Distance)
  - Global Mirror for zSeries (XRC) – DS6000 can be configured as an XRC target only
  - Metro/Global Mirror for zSeries (three-site solution using Synchronous PPRC and XRC) – DS6000 can be configured as an XRC target only

Figure 3-26 The IBM TotalStorage Resiliency Family

The IBM TotalStorage Resiliency Family

The IBM TotalStorage Resiliency Family is a set of products and features that are designed to help you implement storage solutions that keep your business running 24 hours a day, 7 days a week.

These hardware and software features, products, and services are available on the IBM TotalStorage DS6000 and DS8000 series and IBM TotalStorage ESS Models 750 and 800. In addition, a number of advanced Copy Services features that are part of the IBM TotalStorage Resiliency family are available for the DS6000 and DS8000 series. The IBM TotalStorage DS Family also offers systems to support enterprise-class data backup and disaster recovery capabilities. As part of the IBM TotalStorage Resiliency Family of software, IBM TotalStorage FlashCopy point-in-time copy capabilities back up data in the background while allowing users nearly instant access to information on both source and target volumes. Metro and Global Mirror capabilities create duplicate copies of application data at remote sites. High-speed data transfers help to back up data for rapid retrieval.

Copy Services

Copy Services is a collection of functions that provides disaster recovery, data migration, and data duplication functions. Copy Services runs on the DS6000 and DS8000 series and supports open systems and zSeries environments.

Copy Services functions also are supported on the previous generation of storage systems, the IBM TotalStorage Enterprise Storage Server.

Copy Services include the following types of functions:

- FlashCopy, which is a point-in-time copy function
- Remote mirror and copy functions (previously known as Peer-to-Peer Remote Copy or PPRC), which includes:
  - IBM TotalStorage Metro Mirror (previously known as Synchronous PPRC)
  - IBM TotalStorage Global Copy (previously known as PPRC Extended Distance)
  - IBM TotalStorage Global Mirror (previously known as Asynchronous PPRC)
- z/OS Global Mirror (previously known as Extended Remote Copy or XRC)

For information about copy services see also 3.21, "ESS copy services" on page 79.

Metro/Global Copy function

The Metro/Global Copy function allows you to cascade a PPRC pair with a PPRC-XD pair such that the PPRC secondary also serves as the PPRC-XD primary. In this configuration, a primary and secondary pair is established with the secondary located in a nearby site, protected from primary site disasters. The secondary volume for the PPRC-XD copy could be located thousands of miles away and would continue to be updated if the original primary location suffered a disaster.

Metro/Global Mirror function

The Metro/Global Mirror function enables a three-site, high availability disaster recovery solution. It combines the capabilities of both Metro Mirror and Global Mirror functions for greater protection against planned and unplanned outages.

3.27 TotalStorage Expert product highlights

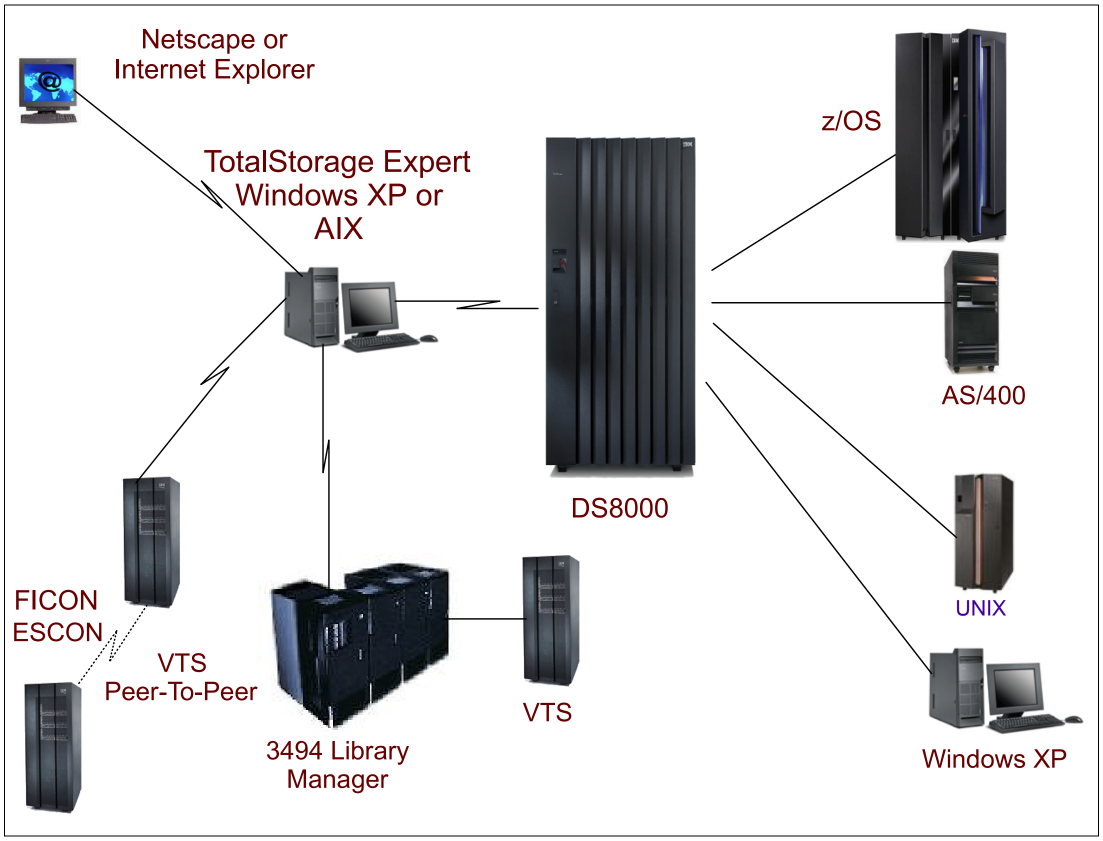


Figure 3-27 TotalStorage Expert

TotalStorage Expert

TotalStorage Expert is an innovative software tool that gives administrators powerful, yet flexible storage asset, capacity, and performance management capabilities to centrally manage Enterprise Storage Servers located anywhere in the enterprise.

IBM TotalStorage Expert has two available features:

- The ESS feature, which supports ESS
- The ETL feature, which supports Enterprise Tape Library products

The two features are licensed separately. There are also upgrade features for users of StorWatch Expert V1 with either the ESS or the ETL feature, or both, who want to migrate to TotalStorage Expert V2.1.1.

TotalStorage Expert is designed to augment commonly used IBM performance tools such as Resource Management Facility (RMF), DFSMS Optimizer, AIX Performance Toolkit, and similar host-based performance monitors. While these tools provide performance statistics from the host system's perspective, TotalStorage Expert provides statistics from the ESS and ETL system perspective.

By complementing other performance tools, TotalStorage Expert provides a more comprehensive view of performance; it gathers and presents information that provides a complete management solution for storage monitoring and administration.

TotalStorage Expert helps storage administrators by increasing the productivity of storage resources.

The ESS is ideal for businesses with multiple heterogeneous servers, including zSeries, UNIX, Windows NT®, Windows 2000, Novell NetWare, HP/UX, Sun Solaris, and AS/400 servers.

With Version 2.1.1, the TotalStorage ESS Expert is packaged with the TotalStorage ETL Expert. The ETL Expert provides performance, asset, and capacity management for IBM's three ETL solutions:

- IBM TotalStorage Enterprise Automated Tape Library, described in "IBM TotalStorage Enterprise Automated Tape Library 3494" on page 103.
- IBM TotalStorage Virtual Tape Server, described in "Introduction to Virtual Tape Server (VTS)" on page 105.
- IBM TotalStorage Peer-to-Peer Virtual Tapeserver, described in "IBM TotalStorage Peer-to-Peer VTS" on page 108.

Both tools can run on the same server, share a common database, efficiently monitor storage resources from any location within the enterprise, and provide a similar look and feel through a Web browser user interface. Together they provide a complete solution that helps optimize the potential of IBM disk and tape subsystems.

3.28 Introduction to tape processing

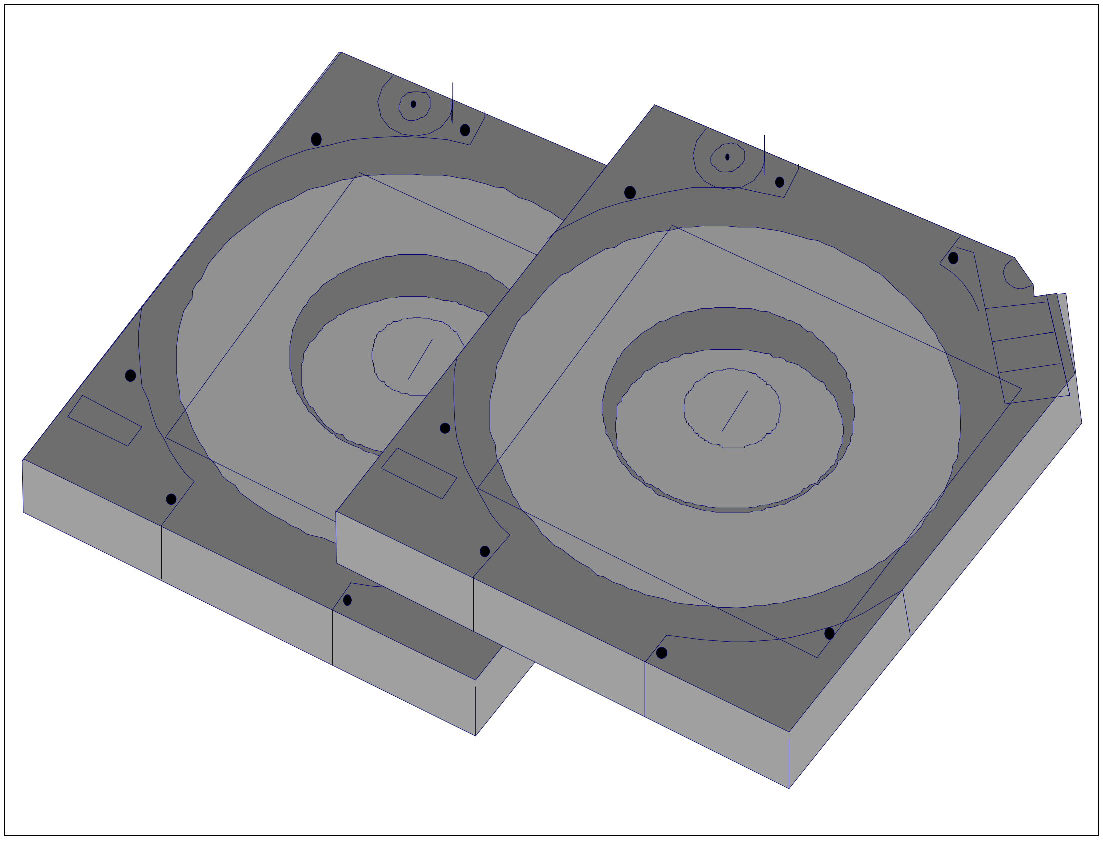


Figure 3-28 Introduction to tape processing

Tape volumes

Tape refer to volumes that can be physically moved. You can only store sequential data sets on tape. Tape volumes can be sent to a safe, or to other data processing centers.

Internal labels are used to identify magnetic tape volumes and the data sets on those volumes. You can process tape volumes with:

- IBM standard labels
- Labels that follow standards published by:
  - International Organization for Standardization (ISO)
  - American National Standards Institute (ANSI)
  - Federal Information Processing Standard (FIPS)
- Nonstandard labels
- No labels

Note: Your installation can install a bypass for any type of label processing; however, the use of labels is recommended as a basis for efficient control of your data.

IBM standard tape labels consist of volume labels and groups of data set labels. The volume label, identifying the volume and its owner, is the first record on the tape. The data set label, identifying the data set and describing its contents, precedes and follows each data set on the volume:

- The data set labels that precede the data set are called *header* labels.
- The data set labels that follow the data set are called *trailer* labels. They are almost identical to the header labels.
- The data set label groups can include standard user labels at your option.

Usually, the formats of ISO and ANSI labels, which are defined by the respective organizations, are similar to the formats of IBM standard labels.

Nonstandard tape labels can have any format and are processed by routines you provide. Unlabeled tapes contain only data sets and tape marks.

3.29 SL and NL format

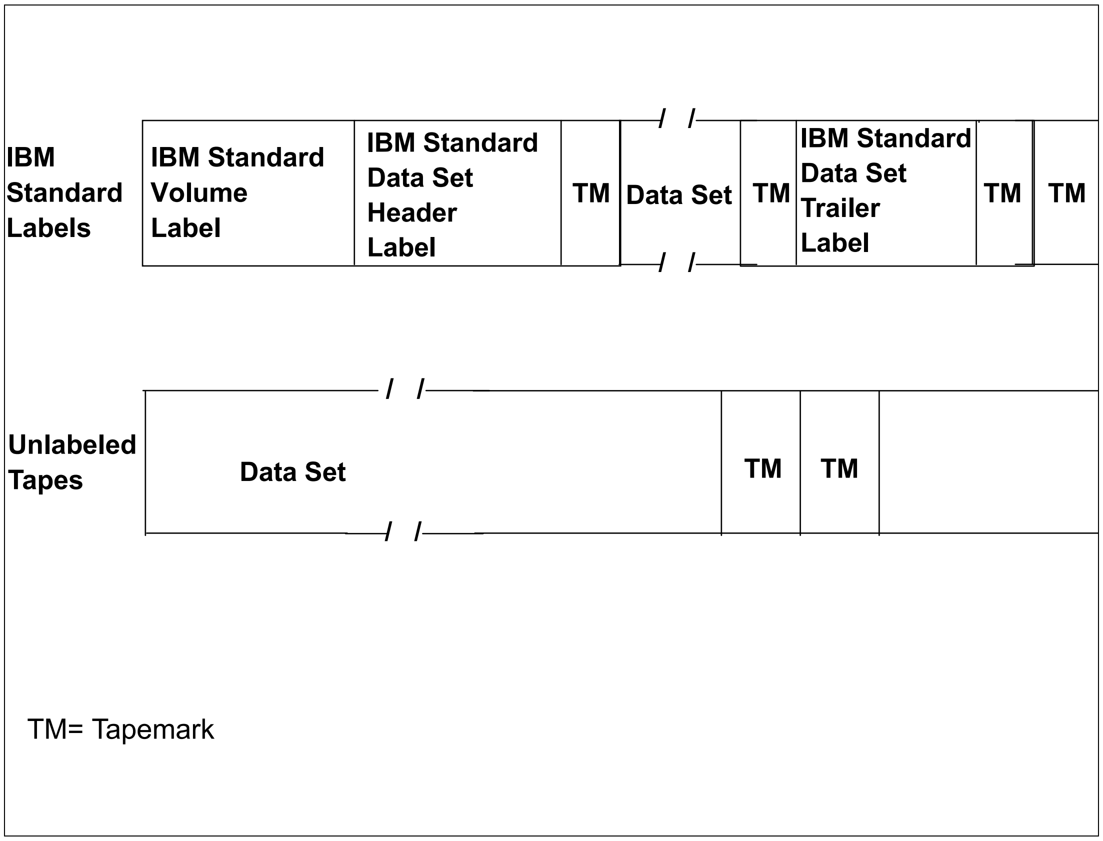

IBM Standard Labels
IBM Standard Volume Label
IBM Standard Data Set Header Label
TM
Data Set
TM
IBM Standard Data Set Trailer Label
TM
TM
Unlabeled Tapes
Data Set
TM
TM
TM= Tapemark

Figure 3-29 SL and NL format

Using tape with JCL

In the job control statements, you must provide a data definition (DD) statement for each data set to be processed. The LABEL parameter of the DD statement is used to describe the data set's labels.

Other parameters of the DD statement identify the data set, give volume and unit information and volume disposition, and describe the data set's physical attributes. You can use a data class to specify all of your data set's attributes (such as record length and record format), but not data set name and disposition. Specify the name of the data class using the JCL keyword DATACLAS. If you do not specify a data class, the automatic class selection (ACS) routines assign a data class based on the defaults defined by your storage administrator.

An example of allocating a tape data set using DATACLAS in the DD statement of the JCL statements follows. In this example, TAPE01 is the name of the data class.

```
//NEW DD DSN=DATASET.NAME,UNIT=TAPE,DISP=(,CATLG,DELETE),DATACLAS=TAPE01,LABEL=(1,SL)
```

Describing the labels

You specify the type of labels by coding one of the subparameters of the LABEL parameter as shown in Table 3-5 on page 98.

Table 3-5 Types of labels

Code	**Meaning**
SL	IBM Standard Label
AL	ISO/ANSI/FIPS labels
SUL	Both IBM and user header or trailer labels
AUL	Both ISO/ANSI/FIPS and user header or trailer labels
NSL	Nonstandard labels
NL	No labels, but the existence of a previous label is verified
BLP	Bypass label processing. The data is treated in the same manner as if NL had been specified, except that the system does not check for an existing volume label. The user is responsible for the positioning. If your installation does not allow BLP, the data is treated exactly as if NL had been specified. Your job can use BLP only if the Job Entry Subsystem (JES) through Job class, RACF through TAPEVOL class, or DFSMSrmm(*) allow it.
LTM	Bypass a leading tape mark. If encountered, on unlabeled tapes from VSE.

Note: If you do not specify the label type, the operating system assumes that the data set has IBM standard labels.

3.30 Tape capacity - tape mount management

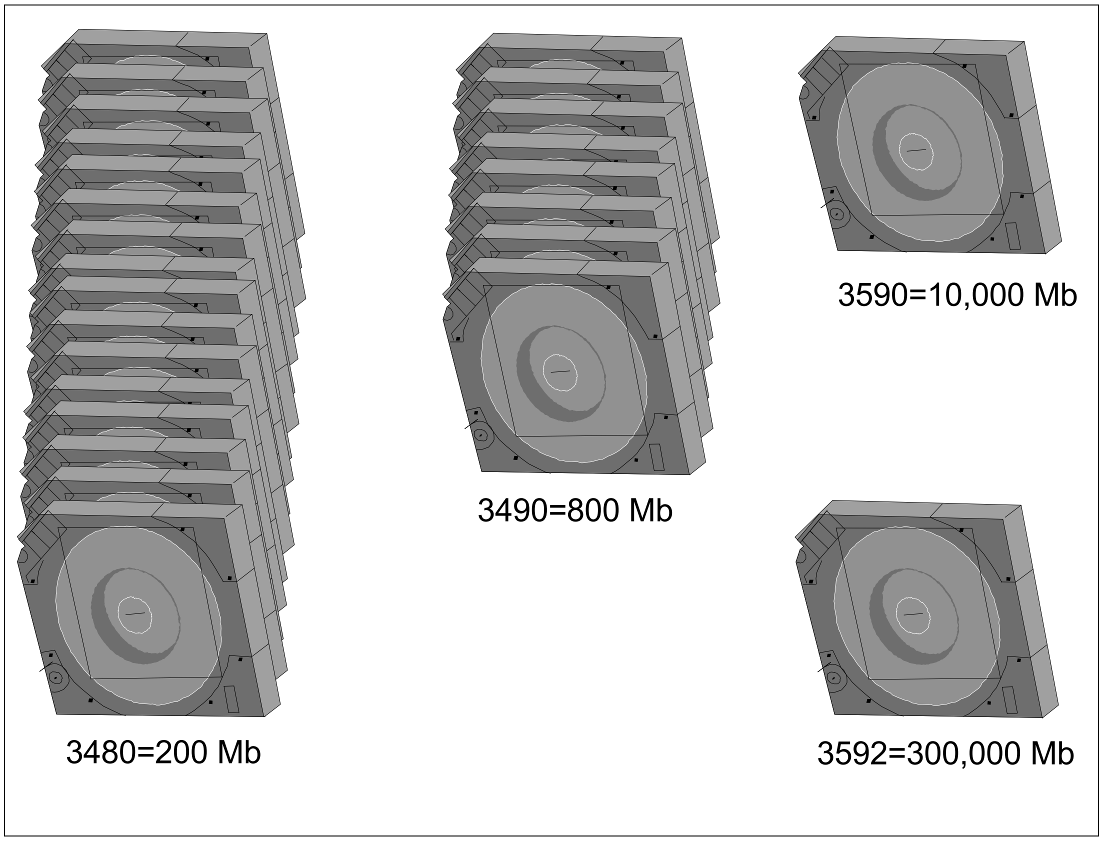


Figure 3-30 Tape capacity

Tape capacity

The capacity of a tape depends on the device type that is recording it. 3480 and 3490 tapes are physically the same cartridges. The IBM 3590 and 3592 high performance cartridge tape is not compatible with the 3480, 3490, or 3490E drives. 3490 units can read 3480 cartridges, but cannot record as a 3480, and 3480 units cannot read or write as a 3490.

Tape mount management

Using DFSMS and tape mount management can help you reduce the number of both tape mounts and tape volumes that your installation requires. The volume mount analyzer reviews your tape mounts and creates reports that provide you with information you need to effectively implement the tape mount management methodology recommended by IBM.

Tape mount management allows you to efficiently fill a tape cartridge to its capacity and gain full benefit from improved data recording capability (IDRC) compaction, 3490E Enhanced Capability Magnetic Tape Subsystem, 36-track enhanced recording format, and Enhanced Capacity Cartridge System Tape. By filling your tape cartridges, you reduce your tape mounts and even the number of tape volumes you need.

With an effective tape cartridge capacity of 2.4 GB using 3490E and the Enhanced Capacity Cartridge System Tape, DFSMS can intercept all but extremely large data sets and manage them with tape mount management. By implementing tape mount management with DFSMS, you might reduce your tape mounts by 60% to 70% with little or no additional hardware required. Therefore, the resulting tape environment would be able to fully exploit integrated cartridge loaders (ICL), IDRC, and 3490E.

Tape mount management also improves job throughput because jobs are no longer queued up on tape drives. Approximately 70% of all tape data sets queued up on drives are less than 10 MB. With tape mount management, these data sets reside on DASD while in use. This frees up the tape drives for other allocations.

Tape mount management recommends that you use DFSMShsm to do interval migration to SMS storage groups. You can use ACS routines to redirect your tape data sets to a tape mount management DASD buffer storage group. DFSMShsm scans this buffer on a regular basis and migrates the data sets to migration level 1 DASD or migration level 2 tape as soon as possible, based on the management class and storage group specifications.

Table 3-6 lists all IBM tape capacities supported since 1952.

Table 3-6 Tape capacity of various IBM products

Year	Product	Capacity (Mb)	Transfer Rate (KB/S)
1952	IBM 726	1.4	7.5
1953	IBM 727	5.8	15
1957	IBM 729	23	90
1965	IBM 2401	46	180
1968	IBM 2420	46	320
1973	IBM 3420	180	1,250
1984	IBM 3480	200	3,000
1989	IBM 3490	200	4,500
1991	IBM 3490E	400	9,000
1992	IBM 3490E	800	9,000
1995	IBM 3590 Magstar	10,000 (uncompacted)	9,000 (uncompacted)
1999	IBM 3590E Magstar	20,000 (uncompacted)	14,000
2000	IBM 3590E Magstar XL Cartridge	20,000/40,000 (dependent on Model B or E)	14,000
2003	IBM 3592 TotalStorage Enterprise Tape Drive	300,000 (for high capacity requirements) or 60,000 (for fast data access requirements)	40,000

For further information about tape processing, see *z/OS DFSMS Using Magnetic Tapes,* SC26-7412.

Note: z/OS supports tape devices starting from D/T 3420.

3.31 TotalStorage Enterprise Tape Drive 3592 Model J1A

- The IBM 3592 is an "Enterprise Class" tape drive
  - Data rate 40 MB/s (without compression)
  - Capacity 300 GB per cartridge (without compression)
  - 60 GB format to provide fast access to data
- Dual ported 2 Gbps Fiber Channel interface
  - Autonegotiates (1 or 2 Gbps, Fabric or loop support)
  - Options may be hard-set at drive
- Drive designed for automation solutions
  - Small form factor
  - Cartridge similar form factor to 3590 and 3490
- Improved environmentals

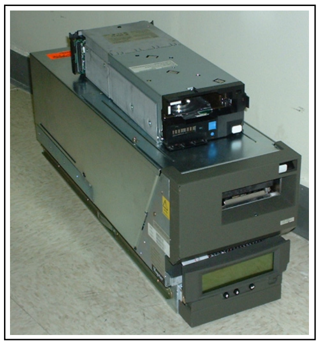


Figure 3-31 TotalStorage Enterprise Tape Drive 3592 Model J1A

IBM 3592 tape drive

The IBM 3592 tape drive is the fourth generation of high capacity and high performance tape systems. It was announced in September 2003 and connects to IBM eServer zSeries systems via the TotalStorage Enterprise Tape Controller 3592 Model J70 using ESCON or FICON links. The 3592 system is the successor of the IBM Magstar 3590 family of tape drives and controller types.

The IBM 3592 tape drive can be used as a standalone solution or as an automated solution within a 3494 tape library.

Enterprise class tape drive

The native rate for data transfer increases up to 40 MB/sec compared to 14 MB/sec in a 3590 Magstar. The uncompressed amount of data which fits on a single cartridge increases to 300 GB and is used for scenarios where high capacity is needed. The tape drive has a second option, where you can store a maximum of 60 GB per tape. This option is used whenever fast access to tape data is needed.

Dual ported 2 Gbps fiber channel interface

This tape drive generation connects to the tape controller 3592 Model J70 via fiber channel. SCSI connection, as it used in 3590 configuration, is no longer supported. However, if you connect a 3590 Magstar tape drive to a 3592 controller, SCSI connection is possible.

Drive designed for automation solutions

The drive has a smaller form factor. Thus, you can integrate more drives into an automated tape library. The cartridges have a similar form factor to the 3590 and 3490 cartridge, so they fit into the same slots in a 3494 automated tape library.

Improved environmentals

By using a smaller form factor than 3590 Magstar drives, you can put two 3592 drives in place of one 3590 drive in the 3494. In a standalone solution you can put a maximum of 12 drives into one 19-inch rack, managed by one controller.

3.32 IBM TotalStorage Enterprise Automated Tape Library 3494

Figure 3-32 3494 tape library

IBM 3494 tape library

Tape storage media can provide low-cost data storage for sequential files, inactive data, and vital records. Because of the continued growth in tape use, *tape automation* has been seen as a way of addressing an increasing number of challenges. Various solutions that provide tape automation are available, including:

- The Automatic Cartridge Loader on IBM 3480 and 3490E tape subsystems, which provides quick scratch (a volume with no valued data, used for output) mount.
- The Automated Cartridge Facility on the Magstar 3590 tape subsystem, which, working with application software, can provide a 10-cartridge mini-tape library.
- The IBM 3494, an automated tape library dataserver, is a device consisting of robotics components, cartridge storage areas (or shelves), tape subsystems, and controlling hardware and software, together with the set of tape volumes that reside in the library and can be mounted on the library tape drives.
- The Magstar Virtual Tape Server (VTS), which provides *volume stacking* capability and exploits the capacity and bandwidth of Magstar 3590 technology.

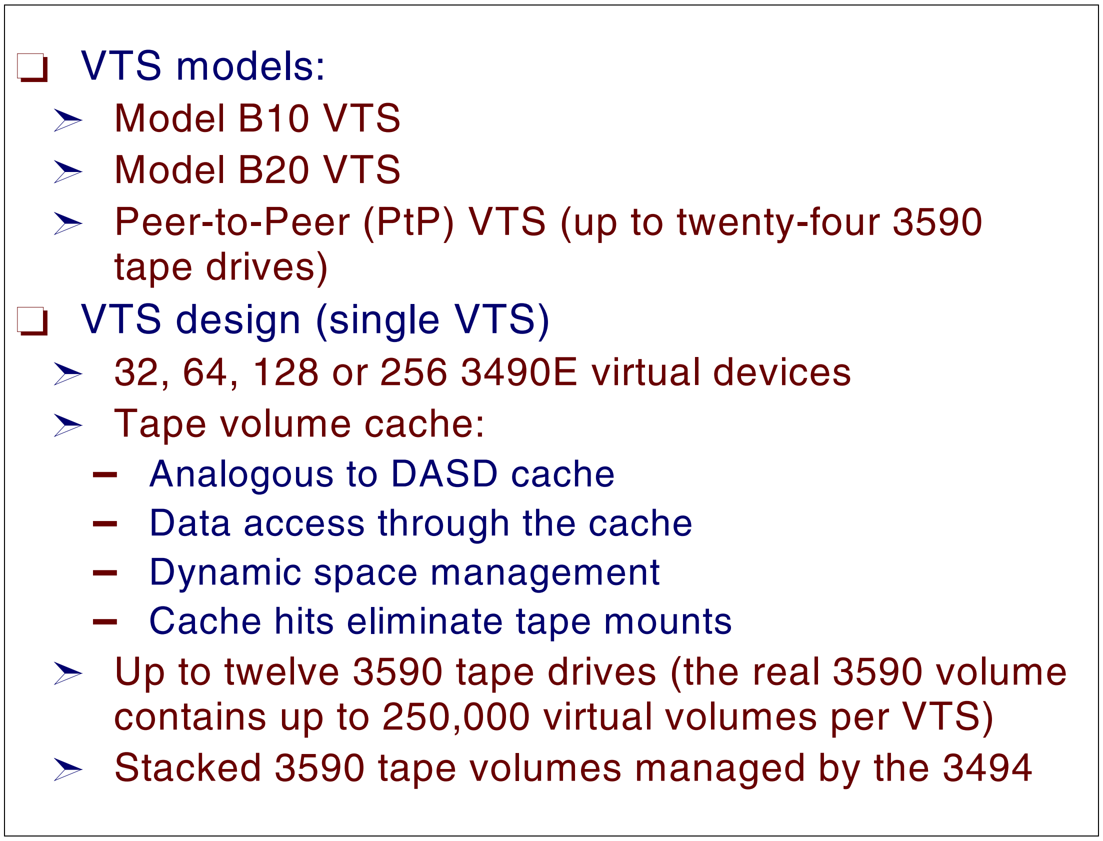
3494 models and features

IBM 3494 offers a wide range of models and features, including the following:

- Up to 96 tape drives

- Support through the Library Control Unit for attachment of up to 15 additional frames, including the Magstar VTS, for a total of 16 frames, not including the High Availability unit
- Cartridge storage capacity of 291 to 6145 tape cartridges
- Data storage capacity of up to 1.84 PB (Petabytes) of uncompacted data and 5.52 PB of compacted data (at a compression rate of 3:1)
- Support for the High Availability unit that provides a high level of availability for tape automation
- Support for the IBM Total Storage Virtual Tape Server
- Support for the IBM Total Storage Peer-to-Peer VTS
- Support for the following tape drives:
  - IBM 3490E Model F1A tape drive
  - IBM 3490E Model CxA tape drives
  - IBM Magstar 3590 Model B1A tape drives
  - IBM Magstar 3590 Model E1A tape drives
  - IBM Magstar 3590 Model H1A tape drives
  - IBM TotalStorage Enterprise Tape Drive 3592 Model J1A
- Attachment to and sharing by multiple host systems, such as IBM eServer zSeries, iSeries, pSeries®, S/390, RS/6000®, AS/400, HP, and Sun processors
- Data paths through FICON, fibre channels, SCSI-2, ESCON, and parallel channels depending on the tape subsystem installed
- Library management commands through RS-232, a local area network (LAN), and parallel, ESCON, and FICON channels

3.33 Introduction to Virtual Tape Server (VTS)



- ❑ VTS models:
  - ➢ Model B10 VTS
  - ➢ Model B20 VTS
  - ➢ Peer-to-Peer (PtP) VTS (up to twenty-four 3590 tape drives)
- ❑ VTS design (single VTS)
  - ➢ 32, 64, 128 or 256 3490E virtual devices
  - ➢ Tape volume cache:
    - – Analogous to DASD cache
    - – Data access through the cache
    - – Dynamic space management
    - – Cache hits eliminate tape mounts
  - ➢ Up to twelve 3590 tape drives (the real 3590 volume contains up to 250,000 virtual volumes per VTS)
  - ➢ Stacked 3590 tape volumes managed by the 3494

Figure 3-33 Introduction to VTS

VTS introduction

The IBM Magstar Virtual Tape Server (VTS), integrated with the IBM Tape Library Dataservers (3494), delivers an increased level of storage capability beyond the traditional storage products hierarchy. The host software sees VTS as a 3490 Enhanced Capability (3490E) Tape Subsystem with associated standard (CST) or Enhanced Capacity Cartridge System Tapes (ECCST). This virtualization of both the tape devices and the storage media to the host allows for transparent utilization of the capabilities of the IBM 3590 tape technology.

Along with introduction of the IBM Magstar VTS, IBM introduced new views of volumes and devices because of the different knowledge about volumes and devices in the host system and the hardware. Using a VTS subsystem, the host application writes tape data to virtual devices. The volumes created by the hosts are called *Virtual Volumes* and are physically stored in a tape volume cache that is built from RAID DASD.

VTS models

These are the IBM 3590 drives you can choose:

- For the Model B10 VTS, four, five, or six 3590-B1A/E1A/H1A can be associated with VTS.
- For the Model B20 VTS, six to twelve 3590-B1A/E1A/H1A can be associated with VTS.

Each ESCON channel in the VTS is capable of supporting 64 logical paths, providing up to 1024 logical paths for Model B20 VTS with sixteen ESCON channels, and 256 logical paths for Model B10 VTS with four ESCON channels. Each logical path can address any of the 32, 64, 128, or 256 virtual devices in the Model B20 VTS.

Each FICON channel in the VTS can support up to 128 logical paths, providing up to 1024 logical paths for the Model B20 VTS with eight FICON channels. With a Model B10 VTS, 512 logical paths can be provided with four FICON channels. As with ESCON, each logical path can address any of the 32, 64, 128, or 256 virtual devices in the Model B20 VTS.

Note: Intermixing FICON and SCSI interfaces is not supported.

Tape volume cache

The IBM TotalStorage Peer-to-Peer Virtual Tape Server appears to the host processor as a single automated tape library with 64, 128, or 256 virtual tape drives and up to 250,000 virtual volumes. The configuration of this system has up to 3.5 TB of Tape Volume Cache native (10.4 TB with 3:1 compression), up to 24 IBM TotalStorage 3590 tape drives, and up to 16 host ESCON or FICON channels.

Through tape volume cache management policies, the VTS management software moves host-created volumes from the tape volume cache to a Magstar cartridge managed by the VTS subsystem. When a virtual volume is moved from the tape volume cache to tape, it becomes a logical volume.

VTS design

VTS looks like an automatic tape library with thirty-two 3490E drives and 50,000 volumes in 37 square feet. Its major components are:

- Magstar 3590 (three or six tape drives) with two ESCON channels
- Magstar 3494 Tape Library
- Fault-tolerant RAID-1 disks (36 Gb or 72 Gb)
- RISC Processor

VTS functions

VTS provides the following functions:

- Thirty-two 3490E virtual devices.
- Tape volume cache (implemented in a RAID-1 disk) that contains virtual volumes.

  The tape volume cache consists of a high performance array of DASD and storage management software. Virtual volumes are held in the tape volume cache when they are being used by the host system. Outboard storage management software manages which virtual volumes are in the tape volume cache and the movement of data between the tape volume cache and physical devices. The size of the DASD is made large enough so that more virtual volumes can be retained in it than just the ones currently associated with the virtual devices.

  After an application modifies and closes a virtual volume, the storage management software in the system makes a copy of it onto a physical tape. The virtual volume remains available on the DASD until the space it occupies reaches a predetermined threshold. Leaving the virtual volume in the DASD allows for fast access to it during subsequent requests. The DASD and the management of the space used to keep closed volumes available is called tape volume cache. Performance for mounting a volume that is in tape volume cache is quicker than if a real physical volume is mounted.
- Up to six 3590 tape drives; the real 3590 volume contains logical volumes. Installation sees up to 50,000 volumes.

- Stacked 3590 tape volumes managed by the 3494. It fills the tape cartridge up to 100%. Putting multiple virtual volumes into a stacked volume, VTS uses all of the available space on the cartridge. VTS uses IBM 3590 cartridges when stacking volumes.

VTS is expected to provide a ratio of 59:1 in volume reduction, with dramatic savings in all tape hardware items (drives, controllers, and robots).

3.34 IBM TotalStorage Peer-to-Peer VTS

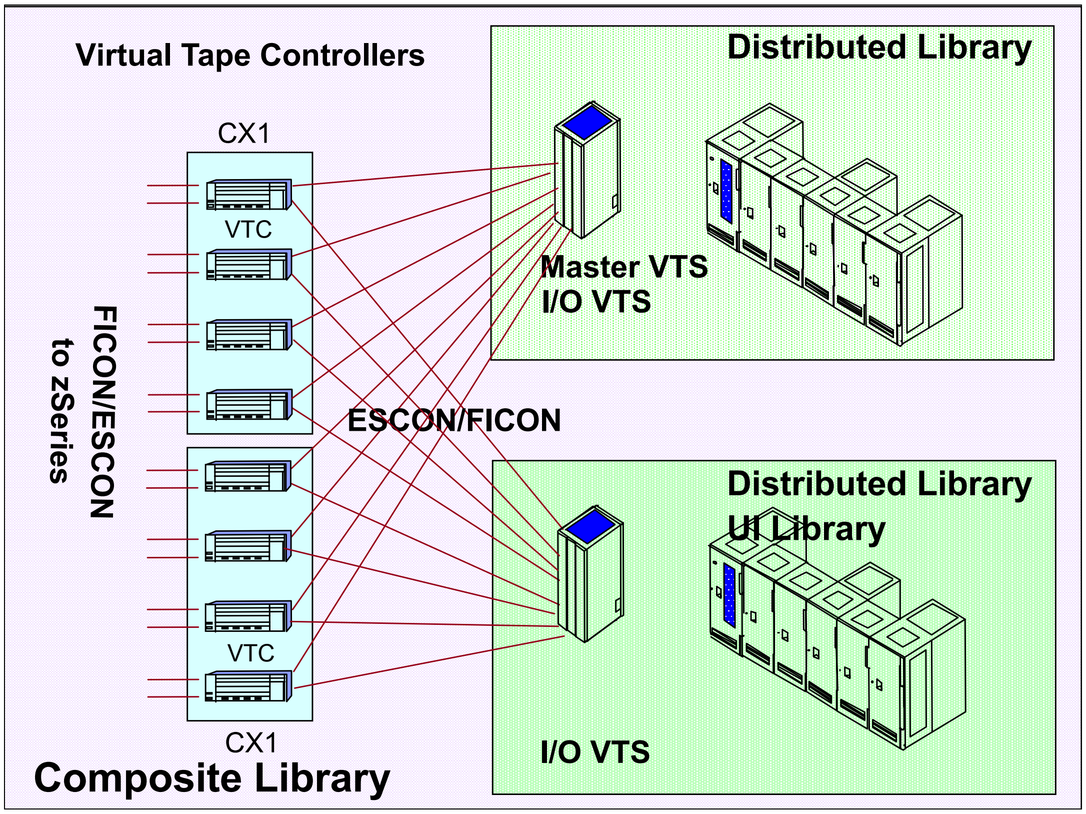


Figure 3-34 IBM TotalStorage Peer-to-Peer VTS

Peer-to-Peer VTS

IBM TotalStorage Peer-to-Peer Virtual Tape Server, an extension of IBM TotalStorage Virtual Tape Server, is specifically designed to enhance data availability. It accomplishes this by providing dual volume copy, remote functionality, and automatic recovery and switchover capabilities. With a design that reduces single points of failure (including the physical media where logical volumes are stored), IBM TotalStorage Peer-to-Peer Virtual Tape Server improves system reliability and availability, as well as data access. To help protect current hardware investments, existing IBM TotalStorage Virtual Tape Servers can be upgraded for use in this new configuration.

IBM TotalStorage Peer-to-Peer Virtual Tape Server consists of new models and features of the 3494 Tape Library that are used to join two separate Virtual Tape Servers into a single, interconnected system. The two virtual tape systems can be located at the same site or at different sites that are geographically remote. This provides a remote copy capability for remote vaulting applications.

IBM TotalStorage Peer-to-Peer Virtual Tape Server appears to the host IBM eServer zSeries processor as a single automated tape library with 64, 128, or 256 virtual tape drives and up to 500,000 virtual volumes. The configuration of this system has up to 3.5 TB of Tape Volume Cache native (10.4 TB with 3:1 compression), up to 24 IBM TotalStorage 3590 tape drives, and up to 16 host ESCON or FICON channels.

In addition to the 3494 VTS components B10, B18, and B20, the Peer-to-Peer VTS consists of the following components:

- The 3494 virtual tape controller model VTC

  The VTC in the Virtual Tape Frame 3494 Model CX1 provides interconnection between two VTSs with the Peer-to-Peer Copy features, and provides two host attachments for the PtP VTS. There must be four (for the Model B10 or B18) or eight (for the Model B20 or B18) VTCs in a PtP VTS configuration. Each VTC is an independently operating, distributed node within the PtP VTS, which continues to operate during scheduled or unscheduled service of another VTC.

- The 3494 auxiliary tape frame model CX1

  The Model CX1 provides the housing and power for two or four 3494 virtual tape controllers. Each Model CX1 can be configured with two or four Model VTCs. There are two power control compartments, each with its own power cord, to allow connection to two power sources.

Peer-to-Peer copy features

Special features installed on 3494 Models B10, B18, and B20 in a Peer-to-Peer configuration provide automatic copies of virtual volumes. These features can be installed on existing VTS systems to upgrade them to a Peer-to-Peer VTS.

VTS advanced functions

As with a stand-alone VTS, the Peer-to-Peer VTS has the option to install some additional features and enhancements to existing features. These new features are:

- Outboard policy management: Outboard policy management enables the storage administrator to manage SMS data classes, storage classes, management classes, and storage groups at the library manager or the 3494 specialist.
- Physical volume pooling: With outboard policy management enabled, you are able to assign logical volumes to selected storage groups. Storage groups point to primary storage pools. These pool assignments are stored in the library manager database. When a logical volume is copied to tape, it is written to a stacked volume that is assigned to a storage pool as defined by the storage group constructs at the library manager.
- Tape volume dual copy: With advanced policy management, storage administrators have the facility to selectively create dual copies of logical volumes within a VTS. This function is also available in the Peer-to-Peer environment. At the site or location where the second distributed library is located, logical volumes can also be duplexed, in which case you could have two or four copies of your data.
- Peer-to-Peer copy control

  There are two types of copy operations:

  - Immediate, which creates a copy of the logical volume in the companion connected virtual tape server prior to completion of a rewind/unload command. This mode provides the highest level of data protection.
  - Deferred, which creates a copy of the logical volume in the companion connected virtual tape server as activity permits after receiving a rewind/unload command. This mode provides protection that is superior to most currently available backup schemes.
- Tape volume cache management: Prior to the introduction of these features, there was no way to influence cache residency. As a result, all data written to the TVC was pre-migrated using a first-in, first-out (FIFO) method. With the introduction of this function, you now have the ability to influence the time that virtual volumes reside in the TVC.

3.35 Storage area network (SAN)

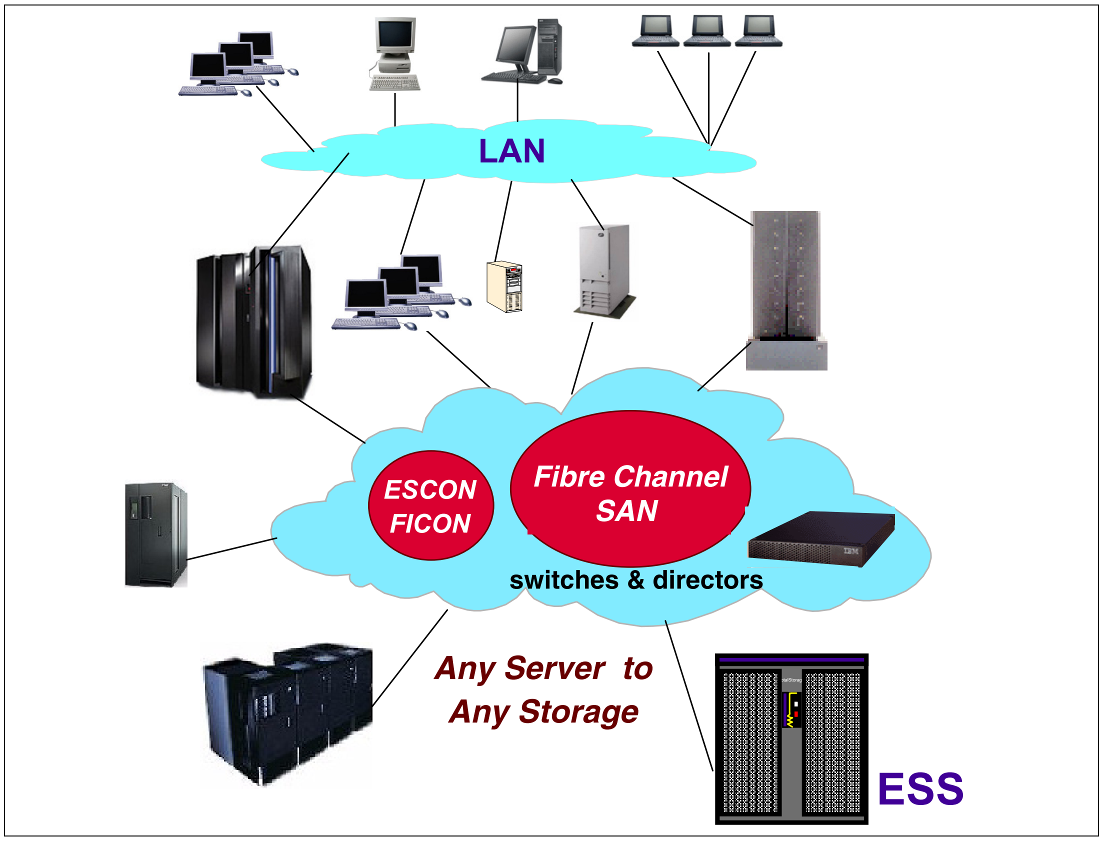


Figure 3-35 Storage area network (SAN)

Storage area network

The Storage Network Industry Association (SNIA) defines a SAN as a network whose primary purpose is the transfer of data between computer systems and storage elements. A SAN consists of a communication infrastructure, which provides physical connections, and a management layer, which organizes the connections, storage elements, and computer systems so that data transfer is secure and robust. The term SAN is usually (but not necessarily) identified with block I/O services rather than file access services. It can also be a storage system consisting of storage elements, storage devices, computer systems, and appliances, plus all control software, communicating over a network.

SANs today are usually built using fibre channel technology, but the concept of a SAN is independent of the underlying type of network.

The major potential benefits of a SAN can be categorized as:

- **Access**

  Benefits include longer distances between processors and storage, higher availability, and improved performance (because I/O traffic is offloaded from a LAN to a dedicated network, and because fibre channel is generally faster than most LAN media). Also, a larger number of processors can be connected to the same storage device, compared to typical built-in device attachment facilities.

- **Consolidation**

  Another benefit is replacement of multiple independent storage devices by fewer devices that support capacity sharing; this is also called *disk and tape pooling*. SANs provide the ultimate in scalability because software can allow multiple SAN devices to appear as a single pool of storage accessible to all processors on the SAN. Storage on a SAN can be managed from a single point of control. Controls over which hosts can see which storage (called *zoning* and *LUN masking*) can be implemented.

- **Protection**

  LAN-free backups occur over the SAN rather than the (slower) LAN, and server-free backups can let disk storage “write itself” directly to tape without processor overhead.

There are different SAN topologies on the base of fibre channel networks:

- **Point-to-Point**

  With a SAN, a simple link is used to provide high-speed interconnection between two nodes.

- **Arbitrated loop**

  The fibre channel arbitrated loop offers relatively high bandwidth and connectivity at a low cost. In order for a node to transfer data, it must first arbitrate to win control of the loop. Once the node has control, it is free to establish a virtual point-to-point connection with another node on the loop. After this point-to-point (virtual) connection is established, the two nodes consume all of the loop’s bandwidth until the data transfer operation is complete. Once the transfer is complete, any node on the loop can then arbitrate to win control of the loop.

- **Switched**

  Fibre channel switches function in a manner similar to traditional network switches to provide increased bandwidth, scalable performance, an increased number of devices, and, in some cases, increased redundancy.

  Multiple switches can be connected to form a switch fabric capable of supporting a large number of host servers and storage subsystems. When switches are connected, each switch’s configuration information has to be copied into all the other participating switches. This is called *cascading*.

FICON and SAN

From a zSeries perspective, FICON is the protocol that is used in a SAN environment. A FICON infrastructure may be point-to-point or switched, using ESCON directors with FICON bridge cards or FICON directors to provide connections between channels and control units. FICON uses fibre channel transport protocols, and so uses the same physical fiber.

Today zSeries has 2 Gbps link data rate support. The 2 Gbps links are for native FICON, FICON CTC, cascaded directors and fibre channels—FCP channels—on the FICON Express cards on z800, z900, and z990 only.

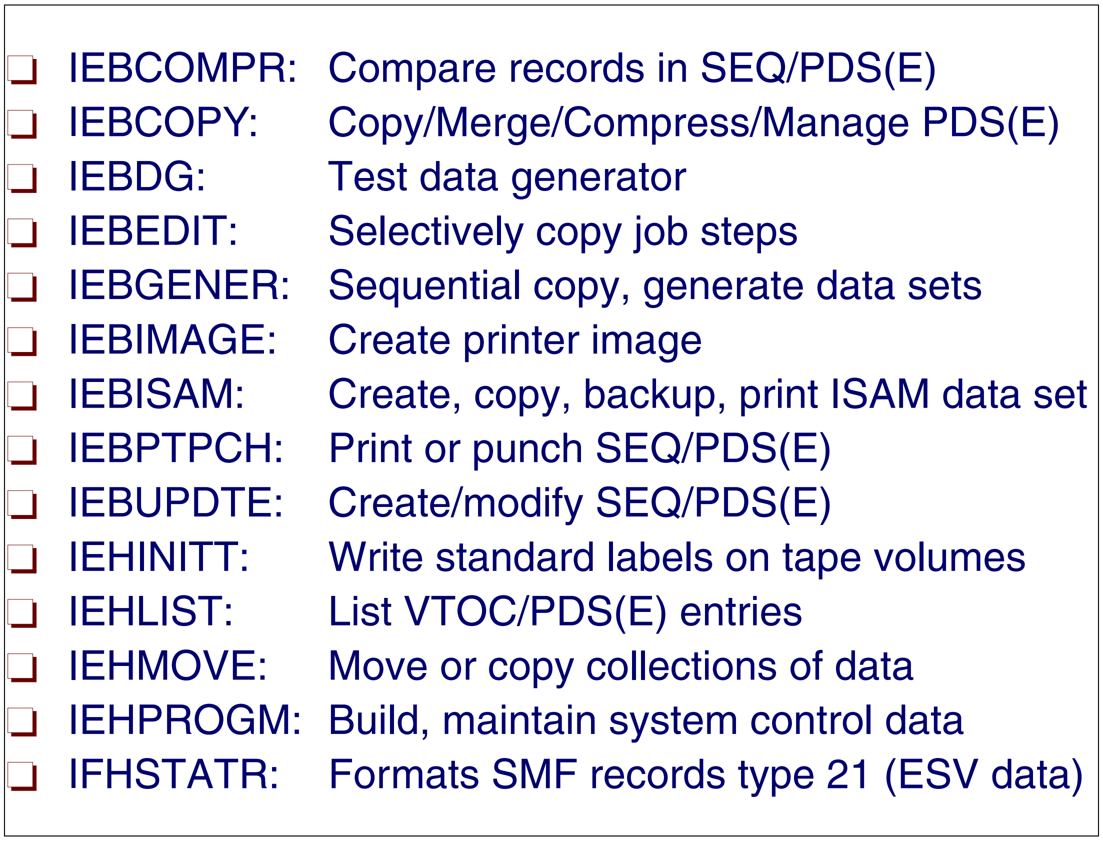
4

Storage management software

DFSMS is an exclusive element of the z/OS operating system and automatically manages data from creation to expiration. In this chapter we present the following:

- The DFSMS utility programs to assist you in organizing and maintaining data
- The major DFSMS access methods
- The data set organizations
- An overview of the elements that comprise DFSMS:
  - DFSMSdfp, a base element of z/OS
  - DFSMSdss, an optional feature of z/OS
  - DFSMShsm, an optional feature of z/OS
  - DFSMSrmm, an optional feature of z/OS
  - z/OS DFSORT

4.1 Overview of DFSMSdfp utilities



❑ IEBCOMPR:	Compare records in SEQ/PDS(E)
❑ IEBCOPY:	Copy/Merge/Compress/Manage PDS(E)
❑ IEBDG:	Test data generator
❑ IEBEDIT:	Selectively copy job steps
❑ IEBGENER:	Sequential copy, generate data sets
❑ IEBIMAGE:	Create printer image
❑ IEBISAM:	Create, copy, backup, print ISAM data set
❑ IEBPTPCH:	Print or punch SEQ/PDS(E)
❑ IEBUPDTE:	Create/modify SEQ/PDS(E)
❑ IEHINITT:	Write standard labels on tape volumes
❑ IEHLIST:	List VTOC/PDS(E) entries
❑ IEHMOVE:	Move or copy collections of data
❑ IEHPROGM:	Build, maintain system control data
❑ IFHSTATR:	Formats SMF records type 21 (ESV data)

Figure 4-1 DFSMSdfp utilities

DFSMSdfp utilities

Utilities are programs that perform commonly needed functions. DFSMS provides utility programs to assist you in organizing and maintaining data. There are system and data set utility programs that are controlled by JCL, and utility control statements.

The base JCL and some utility control statements necessary to use these utilities are provided in the major discussion of the utility programs in this chapter. For more details and to help you find the program that performs the function you need, see “Guide to Utility Program Functions” in topic 1.1 of *z/OS DFSMSdfp Utilities,* SC26-7414.

System utility programs

System utility programs are used to list or change information related to data sets and volumes, such as data set names, catalog entries, and volume labels. Most functions that system utility programs can perform are accomplished more efficiently with other programs, such as IDCAMS,ISPF/PDF, ISMF, or DFSMSrmm.

Table 4-1 on page 115 lists and describes system utilities. Programs that provide functions which are better performed by newer applications (such as ISMF, ISPF/PDF or DFSMSrmm or DFSMSdss) are marked with an asterisk (*) in the table.

Table 4-1 System utility programs

System utility	Alternate program	Purpose
*IEHINITT	DFSMSrmm EDGINERS	Write standard labels on tape volumes.
IEHLIST	ISMF, PDF 3.4	List system control data.
*IEHMOVE	DFSMSdss, IEBCOPY	Move or copy collections of data.
IEHPROGM	Access Method Services, PDF 3.2	Build and maintain system control data.
*IFHSTATR	DFSMSrmm, EREP	Select, format, and write information about tape errors from the IFASMFDP tape.

Data set utility programs

You can use data set utility programs to reorganize, change, or compare data at the data set or record level. These programs are controlled by JCL statements and utility control statements.

These utilities allow you to manipulate partitioned, sequential or indexed sequential data sets, or partitioned data sets extended (PDSEs), which are provided as input to the programs. You can manipulate data ranging from fields within a logical record to entire data sets. The data set utilities included in this section cannot be used with VSAM data sets. You use the IDCAMS utility to manipulate VSAM data set; refer to “Invoking the IDCAMS utility program” on page 136.

Table 4-2 lists data set utility programs and their use. Programs that provide functions which are better performed by newer applications, such as ISMF or DFSMSrmm or DFSMSdss, are marked with an asterisk (*) in the table.

Table 4-2 Data set utility programs

Data set utility	Use
*IEBCOMPR, SuperC, (PDF 3.12)	Compare records in sequential or partitioned data sets, or PDSEs.
IEBCOPY	Copy, compress, or merge partitioned data sets or PDSEs; add RLD count information to load modules; select or exclude specified members in a copy operation; rename or replace selected members of partitioned data sets or PDSEs.
IEBDG	Create a test data set consisting of patterned data.
IEBEDIT	Selectively copy job steps and their associated JOB statements.
IEBGENER or ICEGENER	Copy records from a sequential data set, or convert a data set from sequential organization to partitioned organization.
*IEBIMAGE	Modify, print, or link modules for use with the IBM 3800 Printing Subsystem, the IBM 3262 Model 5, or the 4284 printer.
*IEBISAM	Unload, load, copy, or print an ISAM data set.
IEBPTPCH or PDF 3.1 or 3.6	Print or punch records in a sequential or partitioned data set.
IEBUPDTE	Incorporate changes to sequential or partitioned data sets, or PDSEs.

4.2 IEBCOMPR utility

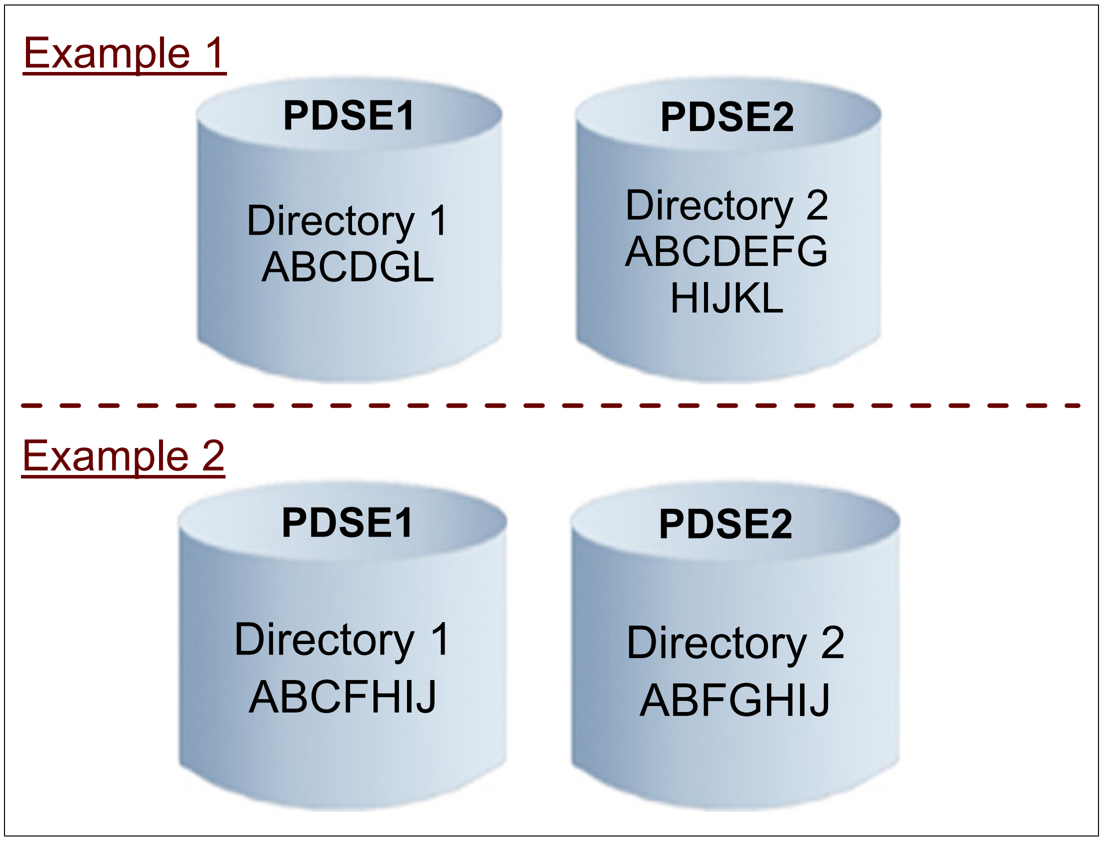


Figure 4-2 IEBCOMPR utility example

IEBCOMPR utility

IEBCOMPR is a data set utility used to compare two sequential data sets, two partitioned data sets (PDS), or two PDSEs, at the logical record level, to verify a backup copy. Fixed, variable, or undefined records from blocked or unblocked data sets or members can also be compared. However, you should not use IEBCOMPR to compare *load modules*.

Two sequential data sets are considered equal (that is, are considered to be identical) if:

- The data sets contain the same number of records
- Corresponding records and keys are identical

Two partitioned data sets or two PDSEs are considered equal if:

- Corresponding members contain the same number of records
- Note lists are in the same position within corresponding members
- Corresponding records and keys are identical
- Corresponding directory user data fields are identical

If all these conditions are not met for a specific type of data set, those data sets are considered *unequal*. If records are unequal, the record and block numbers, the names of the DD statements that define the data sets, and the unequal records are listed in a message data set. Ten successive unequal comparisons stop the job step, unless you provide a routine for handling error conditions.

Note: Load module partitioned data sets that reside on different types of devices should not be compared. Under most circumstances, the data sets will not compare as equal.

A partitioned data set or partitioned data set extended can be compared only if all names in one or both directories have counterpart entries in the other directory. The comparison is made on members identified by these entries and corresponding user data.

Recommendation: Use the SuperC utility instead of IEBCOMPR. SuperC is part of ISPF/PDF and the High Level Assembler Toolkit Feature. SuperC can be processed in the foreground as well as in batch, and its report is more useful.

Examples of comparing data sets

As mentioned, partitioned data sets or PDSEs can be compared only if all the names in one or both of the directories have counterpart entries in the other directory. The comparison is made on members that are identified by these entries and corresponding user data.

You can run this sample JCL to compare two cataloged, partitioned organized (PO) data sets:

```
//DISKDISK  JOB ...
//          EXEC PGM=IEBCOMPR
//SYSPRINT  DD  SYSOUT=A
//SYSUT1    DD  DSN=PDSE1,DISP=SHR
//SYSUT2    DD  DSN=PDSE2,DISP=SHR
//SYSIN     DD  *
          COMPARE TYPORG=PO
/*
```

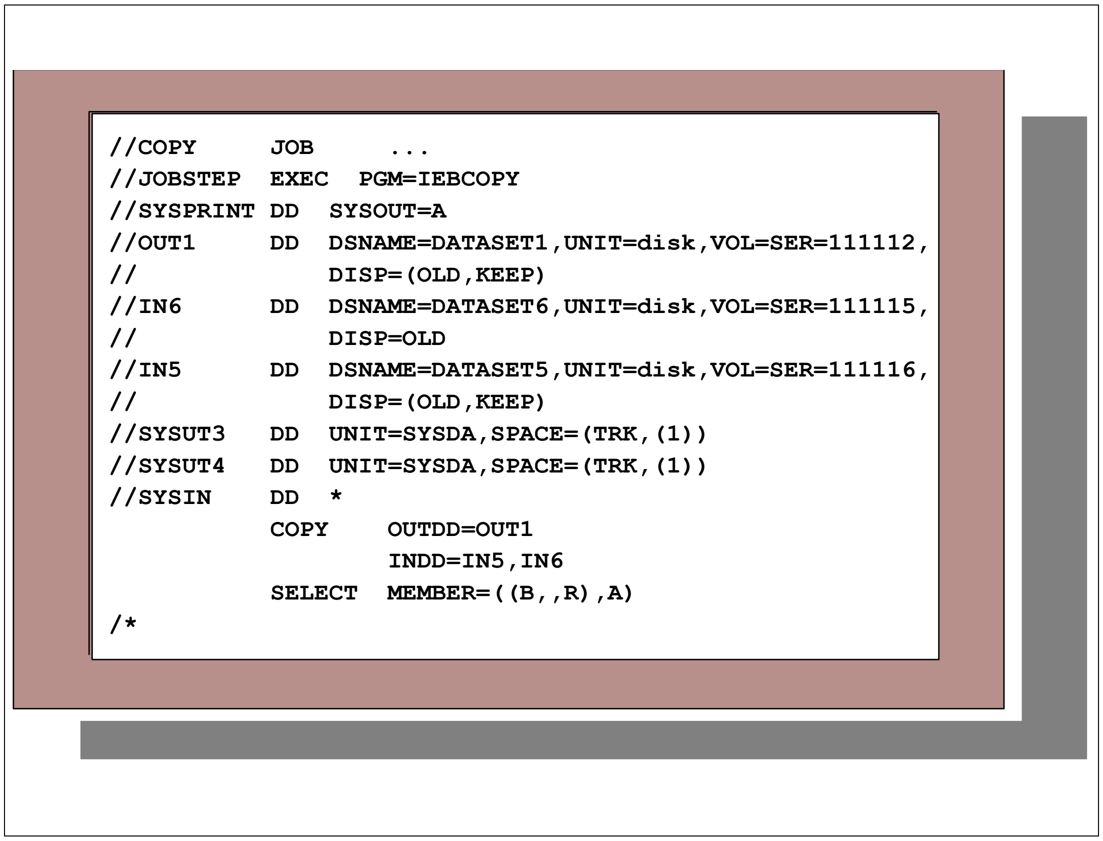
Figure 4-3 IEBCOMPR sample

Figure 4-2 on page 116 shows several examples of the directories of two partitioned data sets.

In Example 1, Directory 2 contains corresponding entries for all the names in Directory 1; therefore, the data sets can be compared.

In Example 2, each directory contains a name that has no corresponding entry in the other directory; therefore, the data sets cannot be compared, and the job step will be ended.

4.3 IEBCOPY utility



```
//COPY      JOB      ...
//JOBSTEP   EXEC  PGM=IEBCOPY
//SYSPRINT DD   SYSOUT=A
//OUT1      DD   DSNAME=DATASET1,UNIT=disk,VOL=SER=111112,
//               DISP=(OLD,KEEP)
//IN6       DD   DSNAME=DATASET6,UNIT=disk,VOL=SER=111115,
//               DISP=OLD
//IN5       DD   DSNAME=DATASET5,UNIT=disk,VOL=SER=111116,
//               DISP=(OLD,KEEP)
//SYSUT3    DD   UNIT=SYSDA,SPACE=(TRK,(1))
//SYSUT4    DD   UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN     DD   *
            COPY    OUTDD=OUT1
                    INDD=IN5,IN6
            SELECT  MEMBER=((B,,R),A)
/*
```

Figure 4-4   IEBCOPY utility example

IEBCOPY utility

IEBCOPY is a data set utility used to copy or merge members between one or more partitioned data sets (PDS), or partitioned data sets extended (PDSE), in full or in part. You can also use IEBCOPY to create a backup of a partitioned data set into a sequential data set (called an unload data set or PDSU), and to copy members from the backup into a partitioned data set.

IEBCOPY is used to:

- Make a copy of a PDS or PDSE
- Merge partitioned data sets (except when unloading)
- Create a sequential form of a PDS or PDSE for a backup or transport
- Reload one or more members from a PDSU into a PDS or PDSE
- Select specific members of a PDS or PDSE to be copied, loaded, or unloaded
- Replace members of a PDS or PDSE
- Rename selected members of a PDS or PDSE
- Exclude members from a data set to be copied, unloaded, or loaded (except on COPYGRP)
- Compress a PDS in place
- Upgrade an OS format load module for faster loading by MVS program fetch

- Copy and re-block load modules
- Convert load modules in a PDS to program objects in a PDSE when copying a PDS to a PDSE
- Convert a PDS to a PDSE, or a PDSE to a PDS
- Copy to or from a PDSE data set a member and its aliases together as a group (COPYGRP)

In addition, IEBCOPY automatically lists the number of unused directory blocks and the number of unused tracks available for member records in the output partitioned data set.

INDD statement

This statement specifies the names of DD statements that locate the input data sets. When an INDD=appears in a record by itself (that is, not with a COPY keyword), it functions as a control statement and begins a new step in the current copy operation.

```
INDD=[(]{DDname|(DDname,R)}[,...][)]
```

R specifies that all members to be copied or loaded from this input data set are to replace any identically named members on the output partitioned data set.

OUTDD statement

This statement specifies the name of a DD statement that locates the output data set.

```
OUTDD=DDname
```

SELECT statement

This statement selects specific members to be processed from one or more data sets by coding a SELECT statement to name the members. Alternatively, all members but a specific few can be designated by coding an EXCLUDE statement to name members not to be processed.

4.4 IEBCOPY: Copy operation

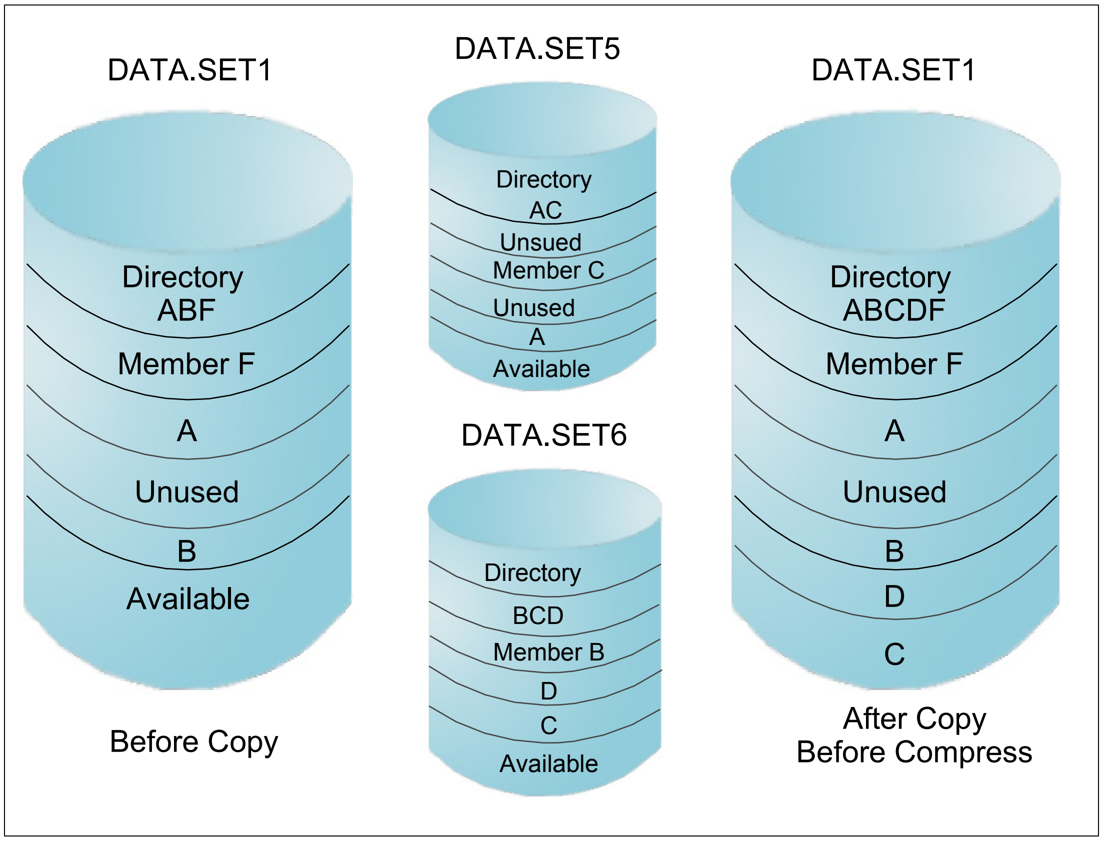


Figure 4-5 IEBCOPY copy operation

Copy control command example

In Figure 4-5, two input partitioned data sets (DATA.SET5 and DATA.SET6) are copied to an existing output partitioned data set (DATA.SET1). In addition, all members on DATA.SET6 are copied; members on the output data set that have the same names as the copied members are replaced. After DATA.SET6 is processed, the output data set (DATA.SET1) is compressed in place. Figure 4-5 shows the input and output data sets before and after copy processing. The compress process is shown in Figure 4-7 on page 122. Figure 4-6 shows the job that is used to copy and compress partitioned data sets.

```
//COPY     JOB  ...
//JOBSTEP  EXEC PGM=IEBCOPY
//SYSPRINT DD   SYSOUT=A
//INOUT1   DD   DSNAME=DATA.SET1,UNIT=disk,VOL=SER=111112,DISP=(OLD,KEEP)
//IN5      DD   DSNAME=DATA.SET5,UNIT=disk,VOL=SER=111114,DISP=OLD
//IN6      DD   DSNAME=DATA.SET6,UNIT=disk,VOL=SER=111115,
//              DISP=(OLD,KEEP)
//SYSUT3   DD   UNIT=SYSDA,SPACE=(TRK,(1))
//SYSUT4   DD   UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN    DD   *
           COPY  OUTDD=INOUT1,INDD=(IN5,(IN6,R),INOUT1)
/*
```

Figure 4-6 IEBOPY with copy and compress

COPY control statement

In the control statement, note the following:

- INOUT1 DD defines a partitioned data set (DATA.SET1) that contains three members (A, B, and F).
- IN5 DD defines a partitioned data set (DATA.SET5) that contains two members (A and C).
- IN6 DD defines a partitioned data set (DATA.SET6), that contains three members (B, C, and D).
- SYSUT3 and SYSUT4 DD define temporary spill data sets. One track is allocated for each DD statement on a disk volume.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPY statement.
- COPY indicates the start of the copy operation. The OUTDD operand specifies DATA.SET1 as the output data set.

COPY processing

Processing occurs as follows:

1. Member A is not copied from DATA.SET5 into DATA.SET1 because it already exists on DATA.SET1 and the replace option was not specified for DATA.SET5.
2. Member C is copied from DATA.SET5 to DATA.SET1, occupying the first available space.
3. All members are copied from DATA.SET6 to DATA.SET1, immediately following the last member. Members B and C are copied even though the output data set already contains members with the same names because the *replace* option is specified on the data set level.

The pointers in DATA.SET1's directory are changed to point to the new members B and C. Thus, the space occupied by the old members B and C is unused.

4.5 IEBCOPY: Compress operation

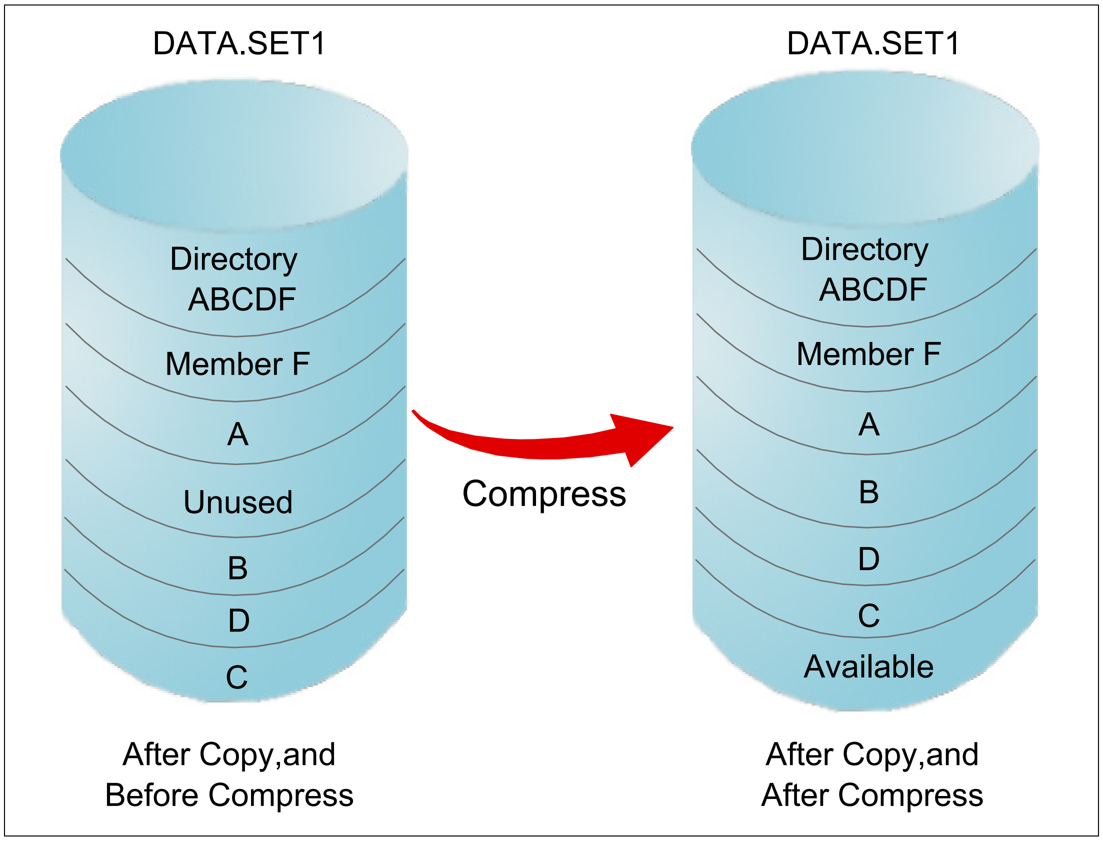


Figure 4-7 IEBCOPY compress operation

IEBCOPY compress operation

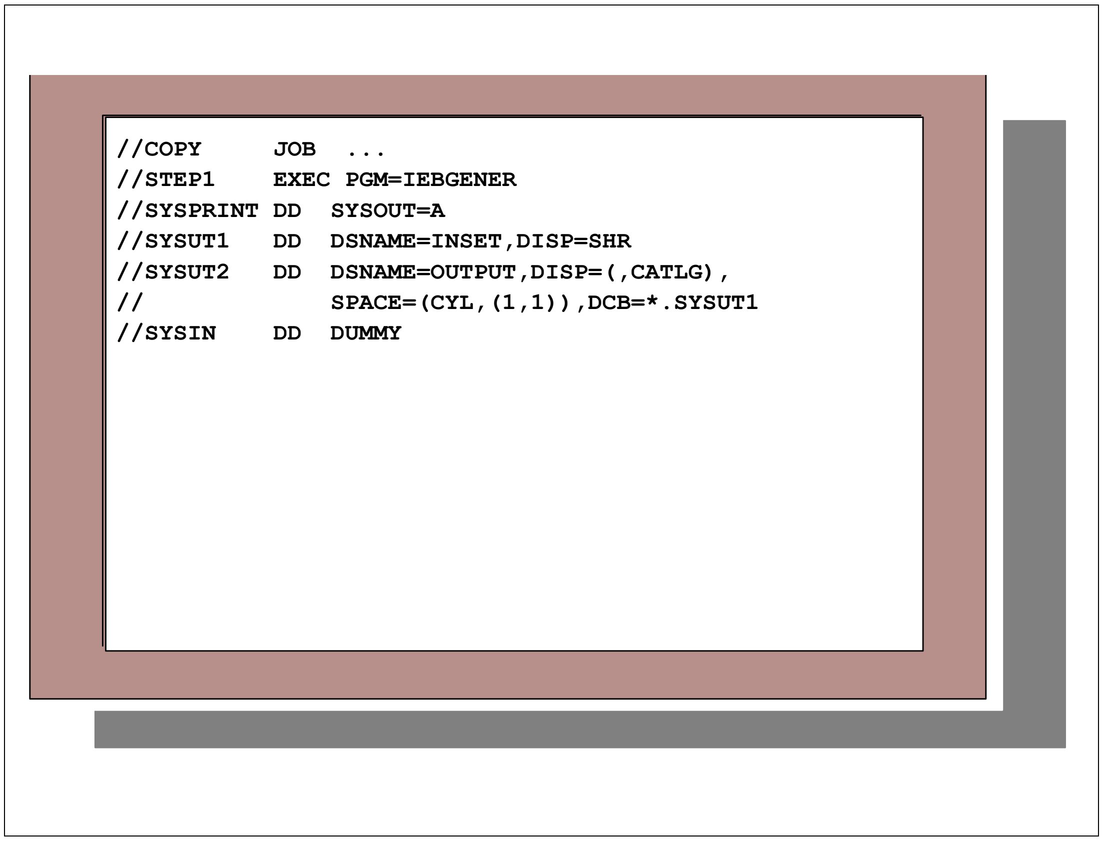
A partitioned data set will contain unused areas (sometimes called gas) where a deleted member or the old version of an updated member once resided. This unused space is only reclaimed when a partitioned data set is copied to a new data set, or after a compress-in-place operation successfully completes. It has no meaning for a PDSE and is ignored if requested.

The simplest way to request a compress-in-place operation is to specify the same ddname for both the OUTDD and INDD parameters of a COPY statement.

Example

In our example in “IEBCOPY: Copy operation” on page 120, the pointers in the DATA.SET1 directory are changed to point to the new members B and C. Thus, the space occupied by the old members B and C is unused. The members currently on DATA.SET1 are compressed in place as a result of the copy operation, thereby eliminating embedded unused space. However, be aware that a compress-in-place operation may bring risk to your data if something abnormally disrupts the process.

4.6 IEBGENER utility



```
//COPY     JOB  ...
//STEP1    EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=INSET,DISP=SHR
//SYSUT2   DD   DSNAME=OUTPUT,DISP=(,CATLG),
//              SPACE=(CYL,(1,1)),DCB=*.SYSUT1
//SYSIN    DD   DUMMY
```

Figure 4-8 IEBGENER utility

Using IEBGENER

IEBGENER copies records from a sequential data set or converts sequential data sets into members of PDSs or PDSEs. You can use IEBGENER to:

- Create a backup copy of a sequential data set, a member of a partitioned data set or PDSE, or a UNIX System Services file such as an HFS file.
- Produce a partitioned data set or PDSE, or a member of a partitioned data set or PDSE, from a sequential data set or a UNIX System Services file.
- Expand an existing partitioned data set or PDSE by creating partitioned members and merging them into the existing data set.
- Produce an edited sequential or partitioned data set or PDSE.
- Manipulate data sets containing double-byte character set data.
- Print sequential data sets or members of partitioned data sets or PDSEs or UNIX System Services files.
- Re-block or change the logical record length of a data set.
- Copy user labels on sequential output data sets.
- Supply editing facilities and exits.

Jobs that call IEBGENER have a system-determined block size used for the output data set if RECFM and LRECL are specified, but BLKSIZE is not specified. The data set is also considered to be system-reblockable.

Copying to z/OS UNIX

IEBGENER can be used to copy from a PDS or PDSE member to a UNIX file. You also can use TSO/E commands and other copying utilities such as ICEGENER or BPXCOPY to copy a PDS or PDSE member to a UNIX file.

In Figure 4-9 on page 124, the data set in SYSUT1 is a PDS or PDSE member and the data set in SYSUT2 is a UNIX file. This job creates a macro library in the UNIX directory.

```
//         JOB ....
//         EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=PROJ.BIGPROG.MACLIB(MAC1),DISP=SHR
//SYSUT2   DD PATH='/u/BIGPROG/macros/special/MAC1',PATHOPTS=OCREAT,
//            PATHDISP=(KEEP,DELETE),
//            PATHMODE=(SIRUSR,SIWUSR,
//            SIRGRP,SIROTH),
//            FILEDATA=TEXT
//SYSIN   DD  DUMMY
```

Figure 4-9 Job to copy a PDS to a z/OS UNIX file

Note: If you have the DFSORT product installed, you should be using ICEGENER as an alternative to IEBGENER when making an unedited copy of a data set or member. It may already be installed in your system under the name IEBGENER. It generally gives better performance.

4.7 IEBGENER: Adding members to a PDS

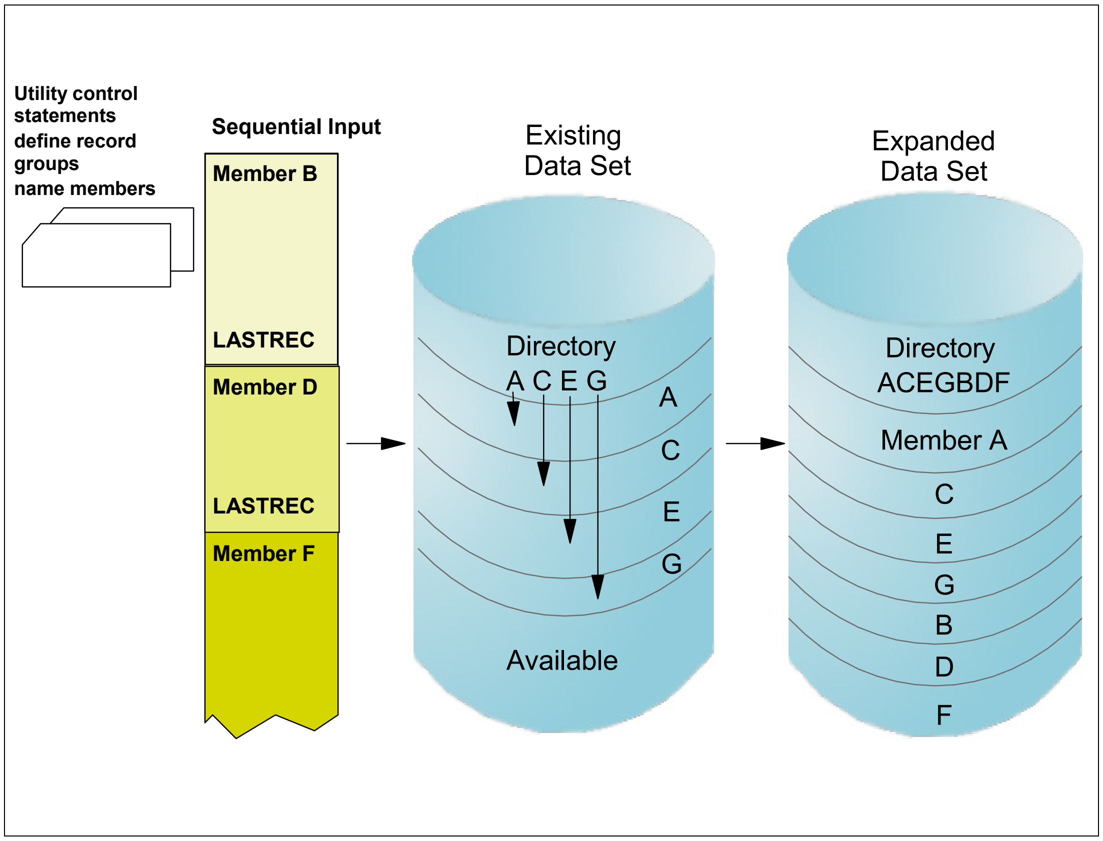


Figure 4-10 Adding members to a PDS using IEBGENER

Adding members to a PDS

You can use IEBGENER to add members to a partitioned data set or PDSE. IEBGENER creates the members from sequential input and adds them to the data set. The merge operation—the ordering of the partitioned directory—is automatically performed by the program.

Figure 4-10 shows how sequential input is converted into members that are merged into an existing partitioned data set or PDSE. The left side of the figure shows the sequential input that is to be merged with the partitioned data set or PDSE shown in the middle of the figure. Utility control statements are used to divide the sequential data set into record groups and to provide a member name for each record group. The right side of the figure shows the expanded partitioned data set or PDSE.

Note that members B, D, and F from the sequential data set were placed in available space and that they are sequentially ordered in the partitioned directory.

4.8 IEBGENER: Copying data to tape

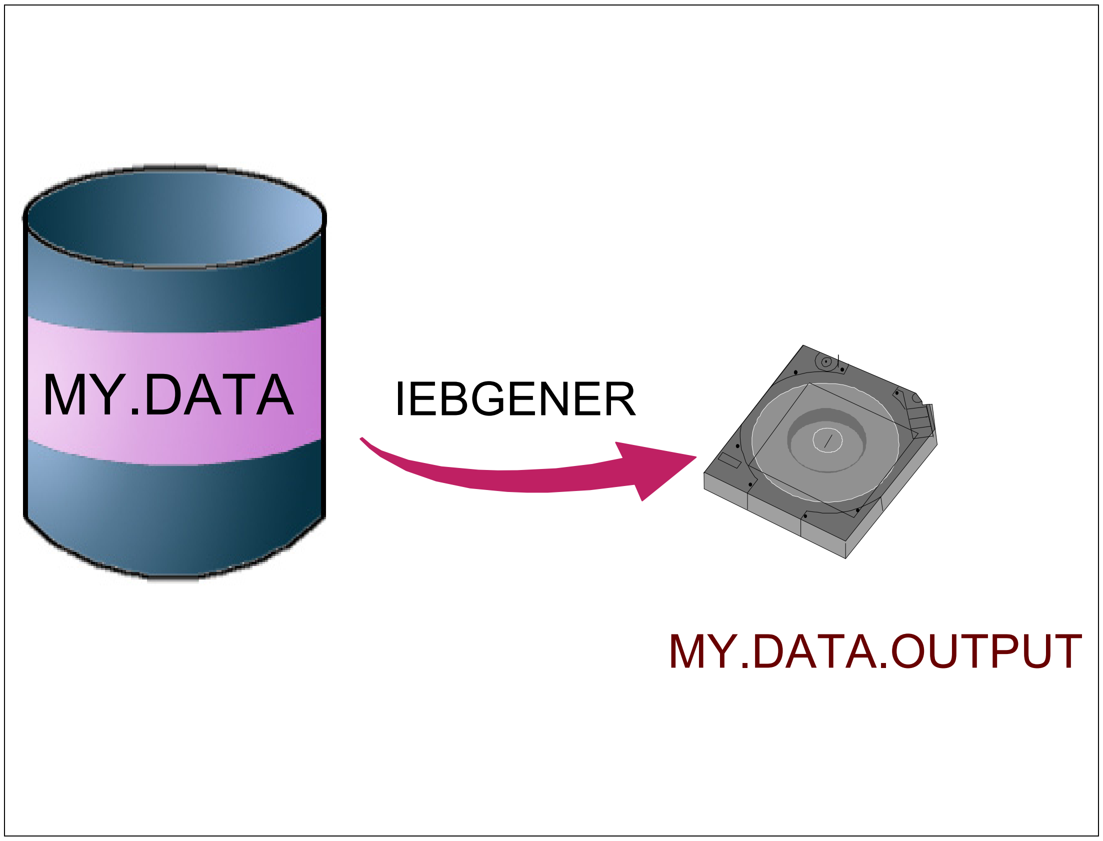


Figure 4-11 Copying data to tape

Copying data to tape example

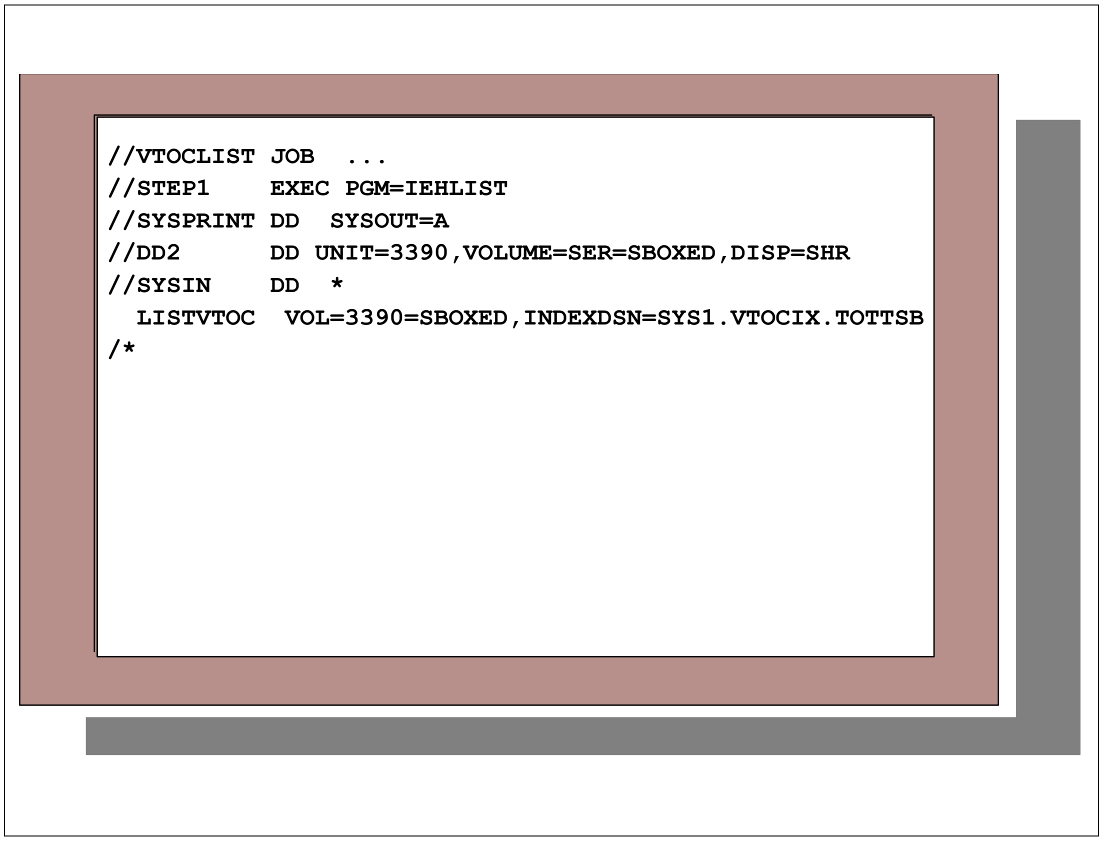
You can use IEBGENER to copy data to tape. The example in Figure 4-11 copies the data set MY.DATA to an SL cartridge. The data set name on tape is MY.DATA.OUTPUT.

```
//DISKTOTP JOB  ...
//STEP1    EXEC PGM=IEBGENER
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSNAME=MY.DATA,DISP=SHR
//SYSUT2   DD  DSNAME=MY.DATA.OUTPUT,UNIT=3490,DISP=(,KEEP),
//             VOLUME=SER=IBM001,LABEL=(1,SL)
//SYSIN    DD   DUMMY
```

Figure 4-12 Copying data to tape with IEBGENER

For further information about IEBGENER, refer to *z/OS DFSMSdfp Utilities,* SC26-7414.

4.9 IEHLIST utility



```
//VTOCLIST JOB  ...
//STEP1    EXEC PGM=IEHLIST
//SYSPRINT DD  SYSOUT=A
//DD2      DD UNIT=3390,VOLUME=SER=SBOXED,DISP=SHR
//SYSIN    DD  *
  LISTVTOC  VOL=3390=SBOXED,INDEXDSN=SYS1.VTOCIX.TOTTSB
/*
```

Figure 4-13 IEHLIST utility

Using IEHLIST

IEHLIST is a system utility used to list entries in the directory of one or more partitioned data sets or PDSEs, or entries in an indexed or non-indexed volume table of contents. Any number of listings can be requested in a single execution of the program.

Listing a PDS or PDSE directory

IEHLIST can list up to ten partitioned data set or PDSE directories at a time.

The directory of a partitioned data set is composed of variable-length records blocked into 256-byte blocks. Each directory block can contain one or more entries that reflect member or alias names and other attributes of the partitioned members. IEHLIST can list these blocks in edited and unedited format.

The directory of a PDSE, when listed, will have the same format as the directory of a partitioned data set.

Listing a volume table of contents (VTOC)

IEHLIST can be used to list, partially or completely, entries in a specified volume table of contents (VTOC), whether indexed or non-indexed. The program lists the contents of selected data set control blocks (DSCBs) in edited or unedited form.

4.10 IEHLIST LISTVTOC output

```
           CONTENTS OF VTOC ON VOL SBOXED  <THIS VOLUME IS NOT SMS MANAGED>
   THERE IS A  2 LEVEL VTOC INDEX
   DATA SETS ARE LISTED IN ALPHANUMERIC ORDER
----------------DATA SET NAME--------------- CREATED  DATE.EXP    FILE TYPE  SM
@LSTPRC@.REXX.SDSF.#MESSG#.$DATASET          2007.116   00.000  PARTITIONED
ADB.TEMP.DB8A.C0000002.AN154059.CHANGES      2007.052   00.000  SEQUENTIAL
ADMIN.DB8A.C0000051.AN171805.CHANGES         2007.064   00.000  SEQUENTIAL
ADMIN.DB8A.C0000062.AN130142.IFF             2007.066   00.000  PARTITIONED
ADMIN.DB8A.C0000090.SMAP.T0001               2007.067   00.000  SEQUENTIAL
ADMIN.DB8A.C0000091.AN153524.CHANGES         2007.067   00.000  SEQUENTIAL
ADMIN.DB8A.C0000092.AN153552.SHRVARS         2007.067   00.000  PARTITIONED
ADMIN.DB8A.C0000116.ULD.T0001                2007.068   00.000  SEQUENTIAL
ADMIN.DB8A.C0000120.SDISC.T0001              2007.068   00.000  SEQUENTIAL
ADMIN.DB8A.C0000123.AN180219.IFF             2007.068   00.000  PARTITIONED
ADMIN.DB8A.C0000157.AN195422.IFF             2007.071   00.000  PARTITIONED
ADMIN.DB8A.C0000204.AN120807.SHRVARS         2007.074   00.000  PARTITIONED
ADMIN.DB8A.C0000231.AN185546.CHANGES         2007.074   00.000  SEQUENTIAL
ADMIN.DB8A.C0000254.SERR.T0001               2007.078   00.000  SEQUENTIAL
BART.CG38V1.EMP4XML.DATA                     2004.271   00.000  SEQUENTIAL
BART.MAZDA.IFCID180.FB80                     2004.254   00.000  SEQUENTIAL
BART.PM32593.D060310.T044716.DMRABSUM.UNL    2006.072   00.000  SEQUENTIAL
BART.PM32593.D060317.T063831.DMRAPSUM.UNL    2006.079   00.000  SEQUENTIAL
BART.PM32593.D060324.T061425.DMRABSUM.UNL    2006.086   00.000  SEQUENTIAL
 THERE ARE    45 EMPTY CYLINDERS PLUS   162  EMPTY TRACKS ON THIS VOLUME
 THERE ARE    4238 BLANK DSCBS IN THE VTOC ON THIS VOLUME
 THERE ARE    301 UNALLOCATED VIRS IN  THE INDEX
```

Figure 4-14 IEHLIST LISTVTOC output

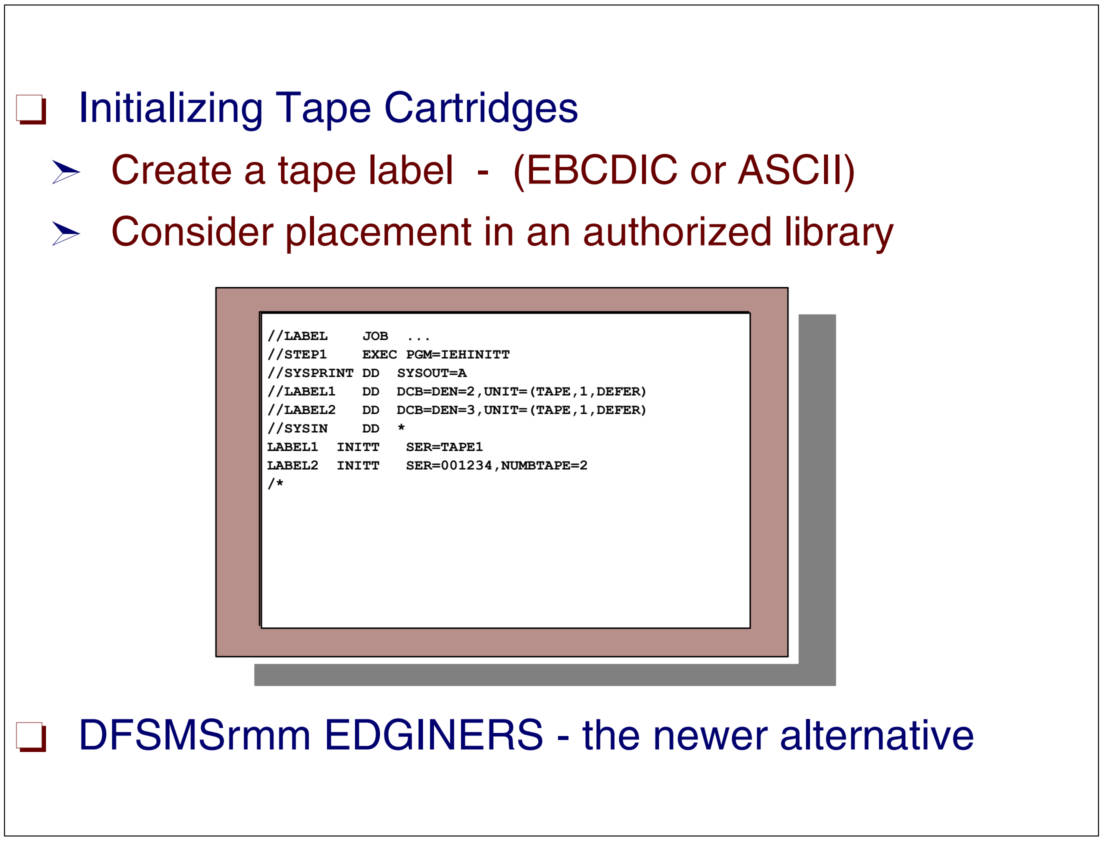
Obtaining the VTOC listing

Running the job shown in Figure 4-13 on page 127 produces a SYSOUT very similar to that shown in Figure 4-14.

If you include the keyword FORMAT in the LISTVTOC parameter, you will have more detailed information about the DASD and about the data sets, and you can also specify the DSNAME that you want to request information about. If you specify the keyword DUMP instead of FORMAT, you will get an unformatted VTOC listing.

Note: This information is at the DASD volume level, and does not have any interaction with the catalog.

4.11 IEHINITT utility



- ❑ Initializing Tape Cartridges
  - ➢ Create a tape label - (EBCDIC or ASCII)
  - ➢ Consider placement in an authorized library

```
//LABEL    JOB  ...
//STEP1    EXEC PGM=IEHINITT
//SYSPRINT DD   SYSOUT=A
//LABEL1   DD   DCB=DEN=2,UNIT=(TAPE,1,DEFER)
//LABEL2   DD   DCB=DEN=3,UNIT=(TAPE,1,DEFER)
//SYSIN    DD   *
LABEL1  INITT   SER=TAPE1
LABEL2  INITT   SER=001234,NUMBTAPE=2
/*
```

- ❑ DFSMSrmm EDGINERS - the newer alternative

Figure 4-15 IEHINITT utility

IEHINITT utility

IEHINITT is a system utility used to place standard volume label sets onto any number of magnetic tapes mounted on one or more tape units. They can be ISO/ANSI Version 3 or ISO/ANSI Version 4 volume label sets written in ASCII (American Standard Code for Information Interchange) or IBM standard labels written in EBCDIC.

Attention: Because IEHINITT can overwrite previously labeled tapes regardless of expiration date and security protection, IBM recommends that the security administrator use PROGRAM protection with the following sequence of RACF commands:

```
RDEFINE PROGRAM IEHINITT ADDMEM('SYS1.LINKLIB'//NODPADCHK) UACC(NONE)
PERMIT IEHINITT CLASS(PROGRAM) ID(users or group) ACCESS(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

(Omit `REFRESH` if you did not have this option active previously.)

IEHINITT should be moved into an authorized password-protected private library, and deleted from SYS1.LINKLIB.

To further protect against overwriting the wrong tape, IEHINITT asks the operator to verify each tape mount.

Examples of IEHINITT

In the example in Figure 4-16, two groups of serial numbers, (001234, 001235, 001236, and 001334, 001335, 001336) are placed on six tape volumes. The labels are written in EBCDIC at 800 bits per inch. Each volume labeled is mounted, when it is required, on a single 9-track tape unit.

```
//LABEL3 JOB  ...
//STEP1    EXEC PGM=IEHINITT
//SYSPRINT DD  SYSOUT=A
//LABEL DD DCB=DEN=2,UNIT=(tape,1,DEFER)
//SYSIN DD  *
LABEL INITT   SER=001234,NUMBTAPE=3
LABEL INITT   SER=001334,NUMBTAPE=3
/*
```

Figure 4-16 IEHINITT example to write EBCDIC labels in different densities

In Figure 4-17, serial numbers 001234, 001244, 001254, 001264, 001274, and so forth are placed on eight tape volumes. The labels are written in EBCDIC at 800 bits per inch. Each volume labeled is mounted, when it is required, on one of four 9-track tape units.

```
//LABEL4   JOB  ...
//STEP1    EXEC PGM=IEHINITT
//SYSPRINT DD  SYSOUT=A
//LABEL    DD  DCB=DEN=2,UNIT=(tape,4,DEFER)
//SYSIN    DD  *
LABEL   INITT   SER=001234
LABEL   INITT   SER=001244
LABEL   INITT   SER=001254
LABEL   INITT   SER=001264
LABEL   INITT   SER=001274
LABEL   INITT   SER=001284
LABEL   INITT   SER=001294
LABEL   INITT   SER=001304
/*
```

Figure 4-17 IEHINITT Place serial number on eight tape volumes

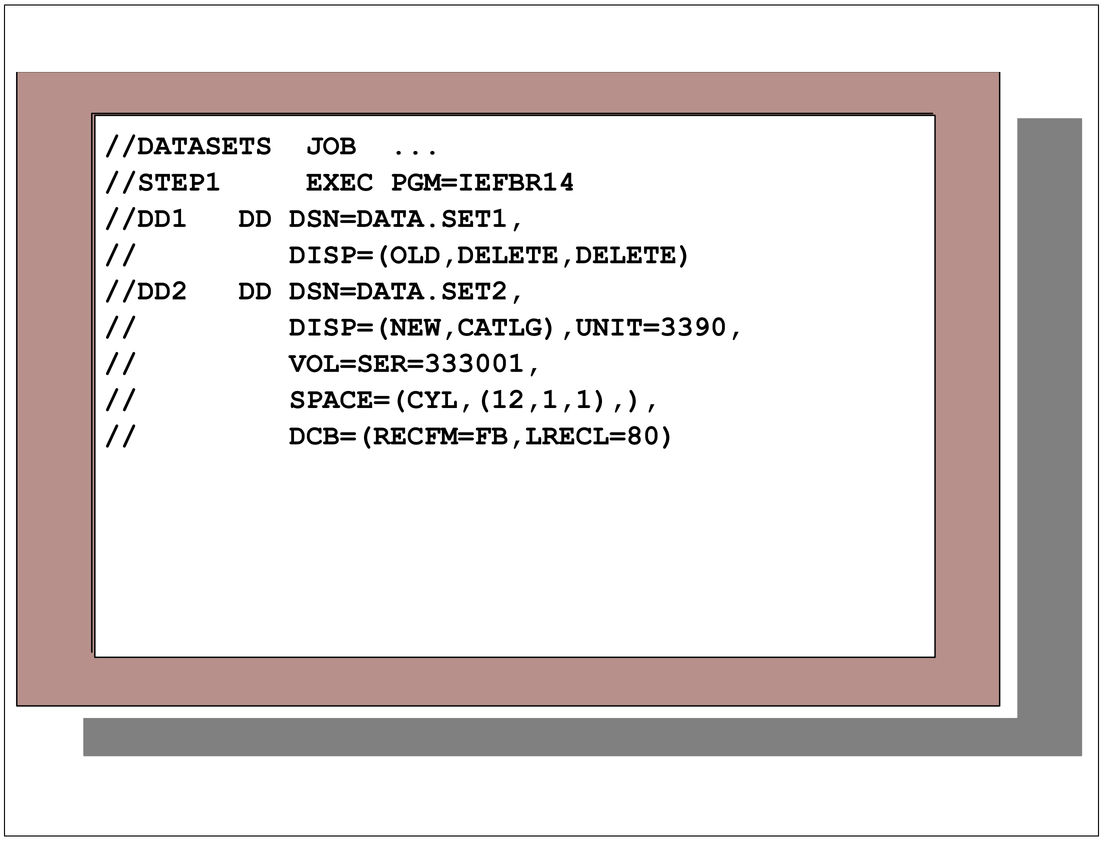
DFSMSrmm EDGINERS utility

The EDGINERS utility program verifies that the volume is mounted before writing a volume label on a labeled, unlabeled, or blank tape. EDGINERS checks security and volume ownership, and provides auditing. DFSMSrmm must know that the volume needs to be labelled. If the labelled volume is undefined, then DFSMSrmm defines it to DFSMSrmm and can create RACF volume security protection.

Detailed procedures for using the program are described in *z/OS DFSMSrmm Implementation and Customization Guide,* SC26-7405.

Note: DFSMSrmm is an optional priced feature of DFSMS. That means that EDGINERS can only be used when DFSMSrmm is licensed. If DFSMSrmm is licensed, IBM recommends that you use EDGINERS for tape initialization instead of using IEHINITT.

4.12 IEFBR14 utility



```
//DATASETS  JOB  ...
//STEP1     EXEC PGM=IEFBR14
//DD1   DD DSN=DATA.SET1,
//           DISP=(OLD,DELETE,DELETE)
//DD2   DD DSN=DATA.SET2,
//           DISP=(NEW,CATLG),UNIT=3390,
//           VOL=SER=333001,
//           SPACE=(CYL,(12,1,1),),
//           DCB=(RECFM=FB,LRECL=80)
```

Figure 4-18 IEFBR14 program

IEFBR14 program

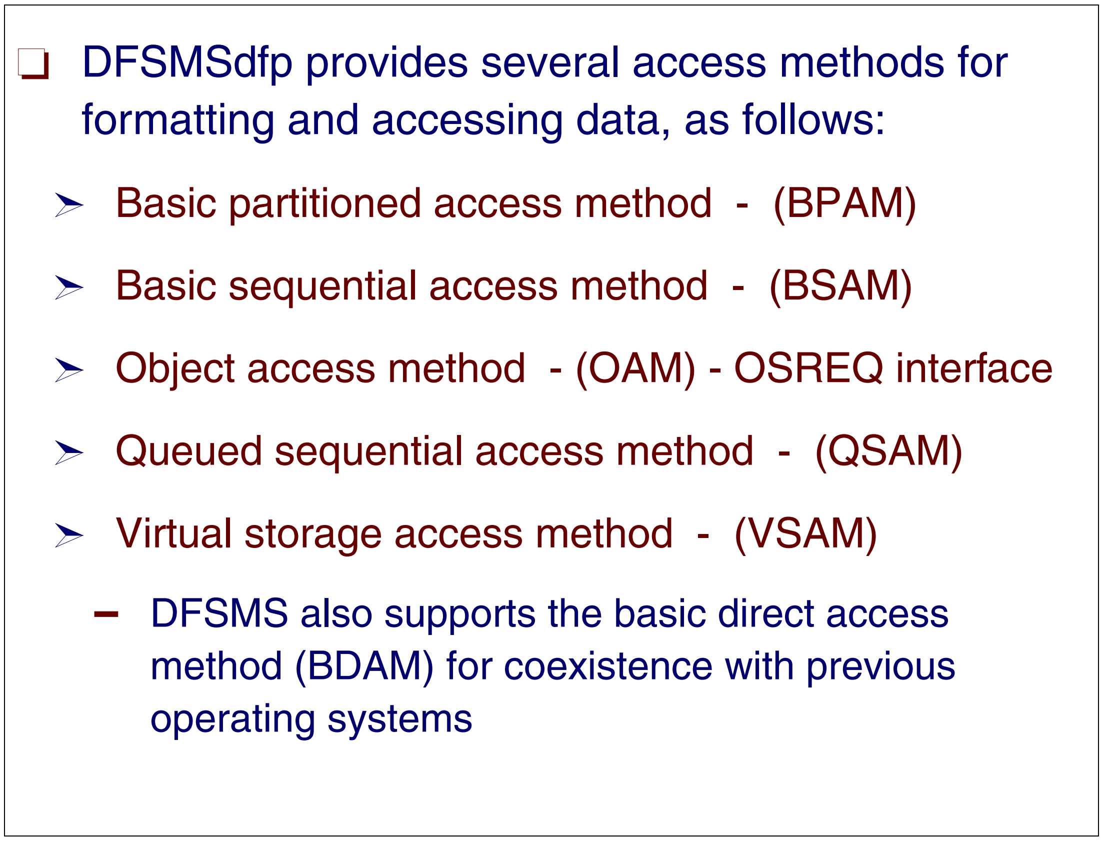
IEFBR14 is not a utility program. It is a two-line program that clears register 15, thus passing a return code of 0. It then branches to the address in register 14, which returns control to the system. So in other words, this program is dummy program. It can be used in a step to force MVS (specifically, the initiator) to process the JCL code and execute functions such as the following:

- Checking all job control statements in the step for syntax
- Allocating direct access space for data sets
- Performing data set dispositions like creating new data sets or deleting old ones

Note: Although the system allocates space for data sets, it does not initialize the new data sets. Therefore, any attempt to read from one of these new data sets in a subsequent step may produce unpredictable results. Also, we do not recommend allocation of multi-volume data sets while executing IEFBR14.

In the example in Figure 4-18 the first DD statement DD1 deletes old data set DATA.SET1. The second DD statement creates a new PDS with name DATA.SET2.

4.13 DFSMSdfp access methods



- DFSMSdfp provides several access methods for formatting and accessing data, as follows:
  - Basic partitioned access method - (BPAM)
  - Basic sequential access method - (BSAM)
  - Object access method - (OAM) - OSREQ interface
  - Queued sequential access method - (QSAM)
  - Virtual storage access method - (VSAM)
    - DFSMS also supports the basic direct access method (BDAM) for coexistence with previous operating systems

Figure 4-19 DFSMSdfp access methods

Access methods

An *access method* is a friendly program interface between programs and their data. It is in charge of interfacing with Input Output Supervisor (IOS), the z/OS code that starts the I/O operation. An access method makes the physical organization of data transparent to you by:

- Managing data buffers
- Blocking and de-blocking logical records into physical blocks
- Synchronizing your task and the I/O operation (wait/post mechanism)
- Writing the channel program
- Optimizing the performance characteristics of the control unit (such as caching and data striping)
- Compressing and decompressing I/O data
- Executing software error recovery

In contrast to other platforms, z/OS supports several types of access methods and data organizations.

An access method defines the organization by which the data is stored and retrieved. DFSMS access methods have their own data set structures for organizing data, macros, and utilities to define and process data sets. It is an application choice, depending on the type of access (sequential or random), to allow or disallow insertions and deletions, to pick up the most adequate access method for its data.

Access methods are identified primarily by the data set organization to which they apply. For example, you can use the basic sequential access method (BSAM) with sequential data sets. However, there are times when an access method identified with one data organization type can be used to process a data set organized in a different manner. For example, a sequential data set (not an extended format data set) created using BSAM can be processed by the basic direct access method (BDAM), and vice versa.

Basic direct access method (BDAM)

BDAM arranges records in any sequence your program indicates, and retrieves records by actual or relative address. If you do not know the exact location of a record, you can specify a point in the data set where a search for the record is to begin. Data sets organized this way are called direct data sets.

Optionally, BDAM uses hardware keys. Hardware keys are less efficient than the optional software keys in VSAM KSDS.

Note: Because BDAM tends to require the use of device-dependent code, it is not a recommended access method. In addition, using keys is much less efficient than in VSAM. BDAM is supported by DFSMS only to enable compatibility with other IBM operating systems.

Basic partitioned access method (BPAM)

BPAM arranges records as members of a partitioned data set (PDS) or a partitioned data set extended (PDSE) on DASD. You can use BPAM to view a UNIX directory and its files as though it were a PDS. You can view each PDS, PDSE, or UNIX member sequentially with BSAM or QSAM. A PDS or PDSE includes a directory that relates member names to locations within the data set. Use the PDS, PDSE, or UNIX directory to retrieve individual members. A member is a sequential file contained in the PDS or PDSE data set. When members contain load modules (executable code in PDS) or program objects (executable code in PDSE), the directory contains program attributes that are required to load and rebind the member. Although UNIX files can contain program objects, program management does not access UNIX files through BPAM.

For information about partitioned organized data set, see 4.22, “Partitioned organized (PO) data sets” on page 148, and subsequent sections.

Basic sequential access method

BSAM arranges logical records sequentially in the order in which they are entered. A data set that has this organization is a sequential data set. Blocking, de-blocking and the I/O synchronization is done by the application program. This is basic access. You can use BSAM with the following data types:

- Sequential data sets
- Extended-format data sets
- z/OS UNIX files

See also 4.28, “Sequential access methods” on page 159.

Queued sequential access method (QSAM)

QSAM arranges logical records sequentially in the order that they are entered to form sequential data sets, which are the same as those data sets that BSAM creates. The system organizes records with other records. QSAM anticipates the need for records based on their order. To improve performance, QSAM reads these records into central storage before they are requested. This is called queued access. QSAM blocks and de-blocks logical records into physical blocks. QSAM also guarantees the synchronization between the task and the I/O operation. You can use QSAM with the same data types as BSAM. See also 4.28, “Sequential access methods” on page 159.

Object Access Method (OAM)

OAM processes very large named byte streams (objects) that have no record boundary or other internal orientation as image data. These objects can be recorded in a DB2 data base or on an optical storage volume. For information about OAM, see *z/OS DFSMS Object Access Method Application Programmer’s Reference,* SC35-0425, and *z/OS DFSMS Object Access Method Planning, Installation, and Storage Administration Guide for Object Support,* SC35-0426.

Virtual Storage Access Method (VSAM)

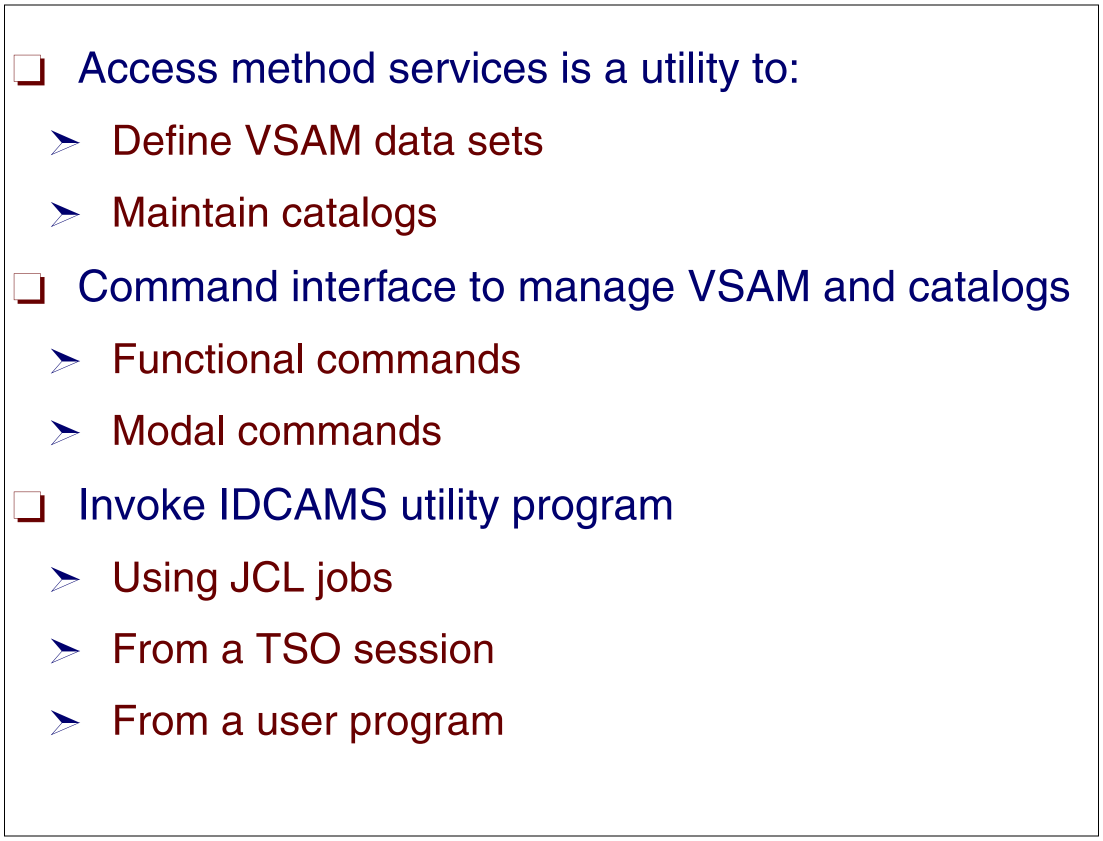
VSAM is an access method that has several ways of organizing data, depending on the application’s needs.

VSAM arranges and retrieves logical records by an index key, relative record number, or relative byte addressing (RBA). A logical record has an RBA, which is the relative byte address of its first byte in relation to the beginning of the data set. VSAM is used for direct, sequential or skip sequential processing of fixed-length and variable-length records on DASD. VSAM data sets (also named clusters) are always cataloged. There are five types of cluster organization:

- *Entry-sequenced data set (ESDS)*: Contains records in the order in which they were entered. Records are added to the end of the data set and can be accessed sequentially or randomly through the RBA.
- *Key-sequenced data set (KSDS)*: Contains records in ascending collating sequence of the contents of a logical record field called *key*. Records can be accessed by the contents of such key, or by an RBA.
- *Linear data set (LDS)*: Contains data that has no record boundaries. Linear data sets contain none of the control information that other VSAM data sets do. Data in Virtual (DIV) is an optional intelligent buffering technique that includes a set of assembler macros that provide buffering access to VSAM linear data sets. See “VSAM: Data-in-virtual (DIV)” on page 177.
- *Relative record data set (RRDS)*: Contains logical records in relative record number order; the records can be accessed sequentially or randomly based on this number. There are two types of relative record data sets:
  - *Fixed-length RRDS*: logical records must be of fixed length.
  - *Variable-length RRDS*: logical records can vary in length.

A z/OS UNIX file (HFS or zFS) can be accessed as though it were a VSAM entry-sequenced data set (ESDS). Although UNIX files are not actually stored as entry-sequenced data sets, the system attempts to simulate the characteristics of such a data set. To identify or access a UNIX file, specify the path that leads to it.

4.14 Access method services (IDCAMS)



- ❑ Access method services is a utility to:
  - ➢ Define VSAM data sets
  - ➢ Maintain catalogs
- ❑ Command interface to manage VSAM and catalogs
  - ➢ Functional commands
  - ➢ Modal commands
- ❑ Invoke IDCAMS utility program
  - ➢ Using JCL jobs
  - ➢ From a TSO session
  - ➢ From a user program

Figure 4-20 Access method services

Access method services

You can use the access method services utility (also know as IDCAMS) to establish and maintain catalogs and data sets (VSAM and non-VSAM). It is used mainly to create and manipulate VSAM data sets. IDCAMS has other functions (such as catalog updates), but it is most closely associated with the use of VSAM.

Access method services commands

There are two types of access method services commands:

Functional commands	Used to request the actual work (for example, defining a data set or listing a catalog)
Modal commands	Allow the conditional execution of functional commands (to make it look like a language)

All access method services commands have the following general structure:

```
COMMAND parameters ... [terminator]
```

The command defines the type of service requested; the parameters further describe the service requested; the terminator indicates the end of the command statement.

Time Sharing Option (TSO) users can use functional commands only. For more information about modal commands, refer to *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

The automatic class selection (ACS) routines (established by your storage administrator) and the associated SMS classes eliminate the need to use many access method services command parameters. The SMS environment is discussed in more detail in “System-managed storage” on page 219.

Invoking the IDCAMS utility program

When you want to use an access method services function, enter a command and specify its parameters. Your request is decoded one command at a time; the appropriate functional routines perform all services required by that command.

You can call the access method services program in the following ways:

- As a job or jobstep
- From a TSO session
- From within your own program

TSO users can run access method services functional commands from a TSO session as though they were TSO commands.

For more information, refer to “Invoking Access Method Services from Your Program” in *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

As a job or jobstep

You can use JCL statements to call access method services. PGM=IDCAMS identifies the access method services program, as shown in Figure 4-21.

```
//YOURJOB  JOB  YOUR INSTALLATION'S JOB=ACCOUNTING DATA
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *

     access method services commands and their parameters

/*
```

Figure 4-21 JCL statements to call IDCAMS

From a TSO session

You can use TSO with VSAM and access method services to:

- Run access method services commands
- Run a program to call access method services

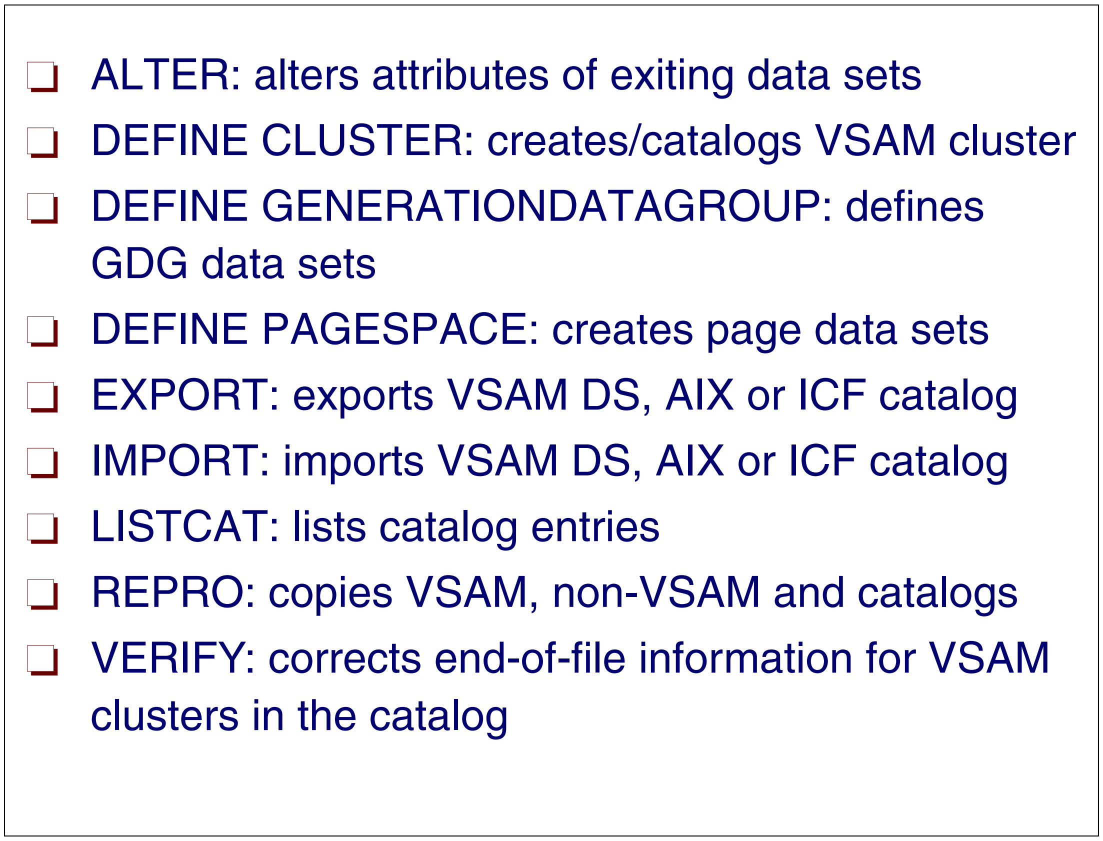
Each time you enter an access method services command as a TSO command, TSO builds the appropriate interface information and calls access method services. You can enter one command at a time. Access method services processes the command completely before TSO lets you continue processing. Except for **ALLOCATE**, all the access method services functional commands are supported in a TSO environment.

To use IDCAMS and some of its parameters from TSO/E, you must update the IKJTSOxx member of SYS1.PARMLIB. Add IDCAMS to the list of authorized programs (AUTHPGM). For more information, refer to *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

From within your own program

You can also call the IDCAMS program from within another program and pass the command and its parameters to the IDCAMS program.

4.15 IDCAMS functional commands



- ALTER: alters attributes of exiting data sets
- DEFINE CLUSTER: creates/catalogs VSAM cluster
- DEFINE GENERATIONDATAGROUP: defines GDG data sets
- DEFINE PAGESPACE: creates page data sets
- EXPORT: exports VSAM DS, AIX or ICF catalog
- IMPORT: imports VSAM DS, AIX or ICF catalog
- LISTCAT: lists catalog entries
- REPRO: copies VSAM, non-VSAM and catalogs
- VERIFY: corrects end-of-file information for VSAM clusters in the catalog

Figure 4-22 Functional commands

IDCAMS functional commands

Table 4-3 lists and describes the functional commands.

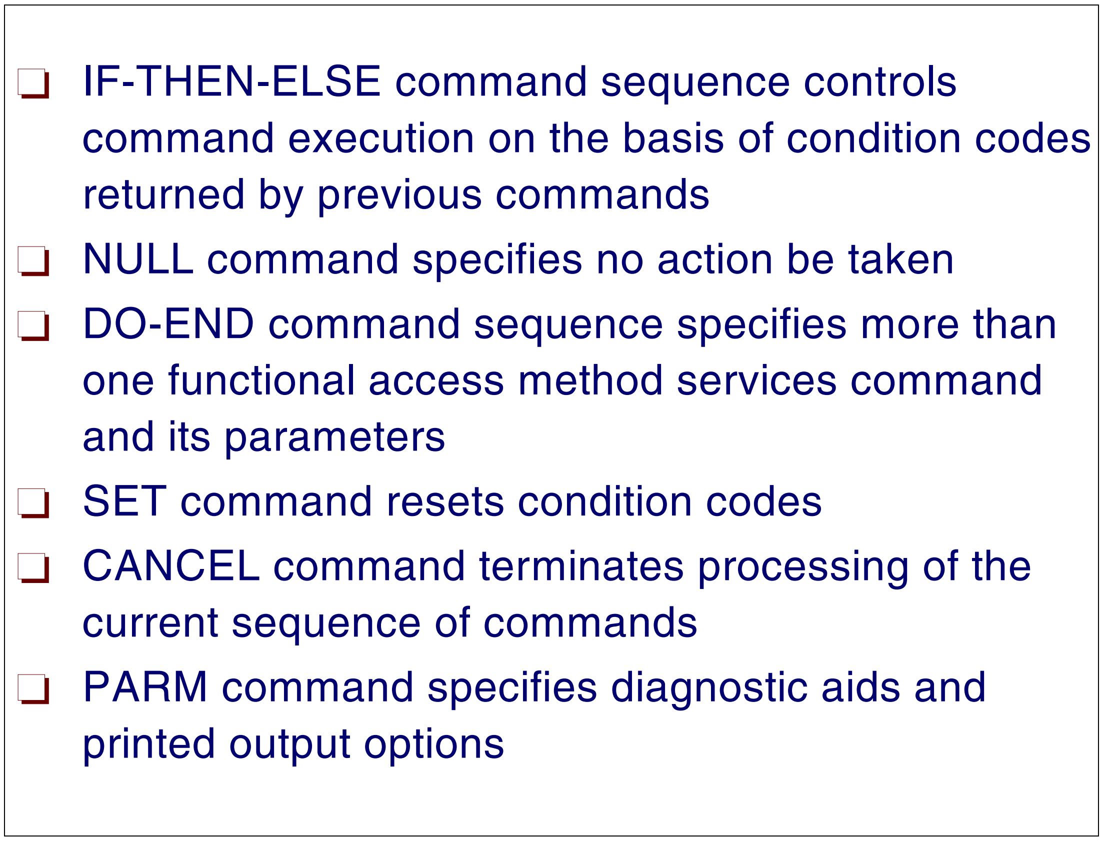
Table 4-3 Functional commands

Command	Functions
ALLOCATE	Allocates VSAM and non-VSAM data sets.
ALTER	Alters attributes of data sets, catalogs, tape library entries, and tape volume entries that have already been defined.
BLDINDEX	Builds alternate indexes (AIX) for existing VSAM base clusters.
CREATE	Creates tape library entries and tape volume entries.
DCOLLECT	Collects data set, volume usage, and migration utility information.
DEFINE ALIAS	Defines an alternate name for a user catalog or a non-VSAM data set.
DEFINE ALTERNATEINDEX	Defines an alternate index for a KSDS or ESDS VSAM data set.
DEFINE CLUSTER	Creates KSDS, ESDS, RRDS, VRRDS and linear VSAM data sets.
DEFINE GENERATIONDATAGROUP	Defines a catalog entry for a generation data group (GDG).
DEFINE NONVSAM	Defines a catalog entry for a non-VSAM data set.

Command	Functions
DEFINE PAGESPACE	Defines an entry for a page space data set.
DEFINE PATH	Defines a path directly over a base cluster or over an alternate index and its related base cluster.
DEFINE USERCATALOG	Defines a user catalog.
DELETE	Deletes catalogs, VSAM clusters, and non-VSAM data sets.
DIAGNOSE	Scans an integrated catalog facility basic catalog structure (BCS) or a VSAM volume data set (VVDS) to validate the data structures and detect structure errors.
EXAMINE	Analyzes and reports the structural consistency of either an index or data component of a KSDS VSAM data set cluster.
EXPORT	Disconnects user catalogs, and exports VSAM clusters and ICF catalog information about the cluster.
EXPORT DISCONNECT	Disconnects a user catalog.
IMPORT	Connects user catalogs, and imports VSAM cluster and its ICF catalogs information.
IMPORT CONNECT	Connects a user catalog or a volume catalog.
LISTCAT	Lists catalog entries.
PRINT	Used to print VSAM data sets, non-VSAM data sets, and catalogs.
REPRO	Performs the following functions: ► Copies VSAM and non-VSAM data sets, user catalogs, master catalogs, and volume catalogs. ► Splits ICF catalog entries between two catalogs. ► Merges ICF catalog entries into another ICF user or master catalog. ► Merges tape library catalog entries from one volume catalog into another volume catalog.
SHCDS	Lists SMSVSAM recovery associated with subsystems spheres and controls that allow recovery of a VSAM RLS environment.
VERIFY	Causes a catalog to correctly reflect the end of a data set after an error occurred while closing a VSAM data set. The error might have caused the catalog to be incorrect.

For a complete description of all AMS commands, refer to *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

4.16 AMS modal commands



- ❑ IF-THEN-ELSE command sequence controls command execution on the basis of condition codes returned by previous commands
- ❑ NULL command specifies no action be taken
- ❑ DO-END command sequence specifies more than one functional access method services command and its parameters
- ❑ SET command resets condition codes
- ❑ CANCEL command terminates processing of the current sequence of commands
- ❑ PARM command specifies diagnostic aids and printed output options

Figure 4-23 AMS modal commands

AMS modal commands

Figure 4-23 contains a list and brief descriptions of the AMS modal commands. With access method services, you can set up jobs to execute a sequence of modal commands with a single invocation of IDCAMS. Modal command execution depends on the success or failure of prior commands. The access method services modal commands are used for the conditional execution of functional commands.

Note: These commands cannot be used when access method services is run in TSO. See *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394, for a complete description of the AMS modal commands.

4.17 DFSMS Data Collection Facility (DCOLLECT)

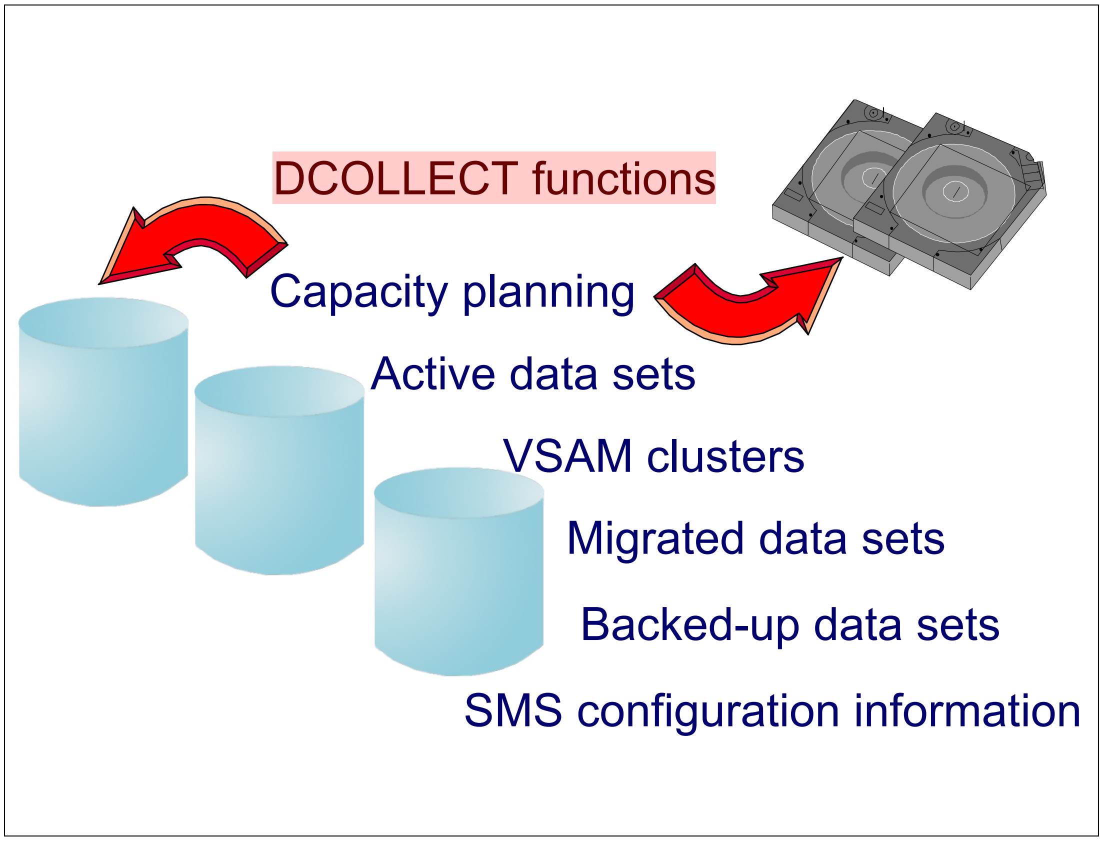


Figure 4-24 Data Collection Facility

DFSMS Data Collection Facility (DCOLLECT)

The DFSMS Data Collection Facility (DCOLLECT) is a function of access method services. The IDCAMS `DCOLLECT` command collects DASD performance and space occupancy data in a sequential file that you can use as input to other programs or applications.

An installation can use this command to collect information about:

- *Active data sets*: DCOLLECT provides data about space use and data set attributes and indicators on the selected volumes and storage groups.
- *VSAM data set information*: DCOLLECT provides specific information relating to VSAM data sets residing on the selected volumes and storage groups.
- *Volumes*: DCOLLECT provides statistics and information about volumes that are selected for collection.
- *Inactive data*: DCOLLECT produces output for DFSMShsm-managed data (inactive data management), which includes both migrated and backed up data sets.
  - Migrated data sets: DCOLLECT provides information about space utilization and data set attributes for data sets migrated by DFSMShsm.
  - Backed up data sets: DCOLLECT provides information about space utilization and data set attributes for every version of a data set backed up by DFSMShsm.
- *Capacity planning*: Capacity planning for DFSMShsm-managed data (inactive data management) includes the collection of both DASD and tape capacity planning.

  - DASD Capacity Planning: DCOLLECT provides information and statistics for volumes managed by DFSMShsm (ML0 and ML1).
  - Tape Capacity Planning: DCOLLECT provides statistics for tapes managed by DFSMShsm.
- *SMS configuration information*: DCOLLECT provides information about the SMS configurations. The information can be from either an active control data set (ACDS) or a source control data set (SCDS), or the active configuration.

Data is gathered from the VTOC, VVDS, and DFSMShsm control data set for both managed and non-managed storage. ISMF provides the option to build the JCL necessary to execute **`DCOLLECT`**.

DCOLLECT example

With the sample JCL Figure 4-25 you can gather information about all volumes belonging to storage group STGGP001.

```
//COLLECT2 JOB    ...
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=A
//OUTDS    DD  DSN=USER.DCOLLECT.OUTPUT,
//             STORCLAS=LARGE,
//             DSORG=PS,
//             DCB=(RECFM=VB,LRECL=644,BLKSIZE=0),
//             SPACE=(1,(100,100)),AVGREC=K,
//             DISP=(NEW,CATLG,KEEP)
//SYSIN    DD     *
           DCOLLECT -
               OFILE(OUTDS) -
               STORAGEGROUP(STGGP001) -
               NODATAINFO
/*
```

Figure 4-25 DCOLLECT job to collect information about all volumes in on storage group

Many customers run DCOLLECT on a time-driven basis triggered by automation. In such a scenario the peak hours should be avoided due to the heavy access rate caused by DCOLLECT in VTOC and catalogs.

4.18 Generation data groups (GDGs)

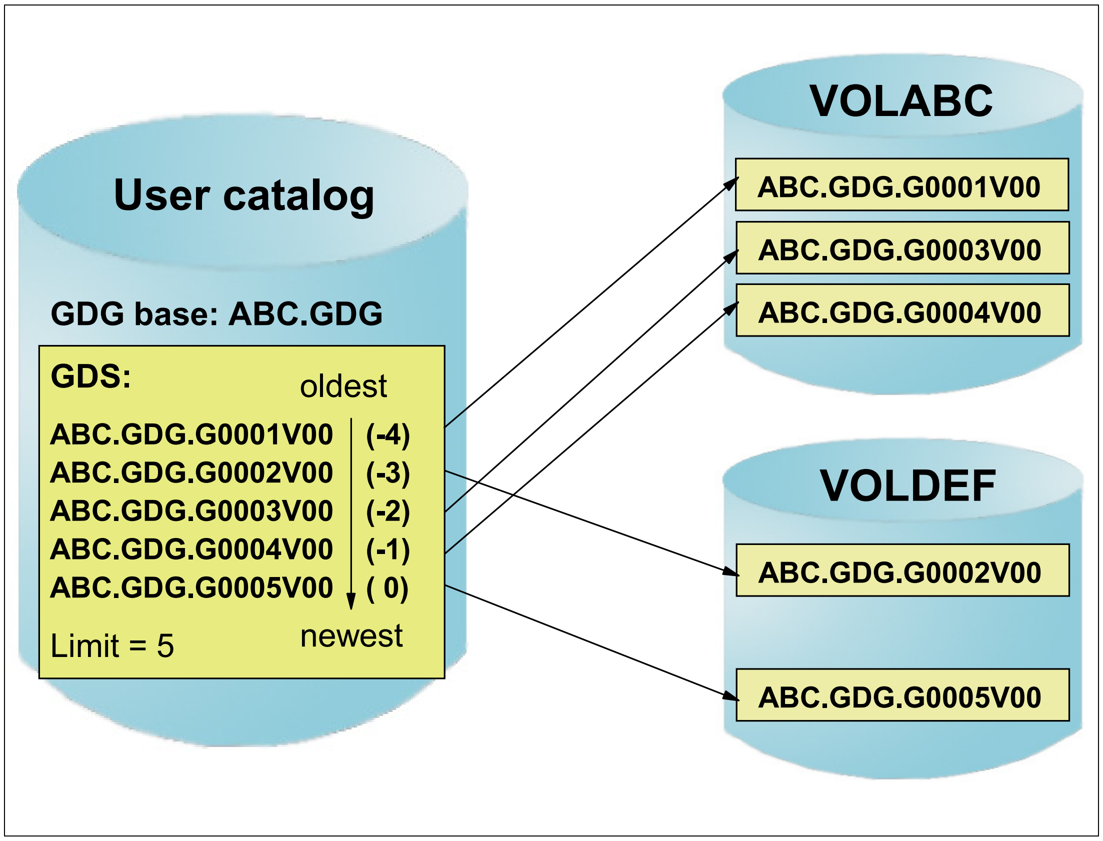


Figure 4-26 Generation data groups (GDG)

Generation data groups

GDG is a catalog function that makes it easier to process data sets with the same type of data but in different update levels. For example, the same data set is produced every day but with different data. Then, you can catalog successive updates or generations of these related data sets. They are called *generation data groups* (GDG). Each data set within a GDG is called a *generation data set* (GDS) or generation. Within a GDG, the generations can have like or unlike DCB attributes and data set organizations. If the attributes and organizations of all generations in a group are identical, the generations can be retrieved together as a single data set.

Generation data sets can be sequential, PDSs, or direct (BDAM). Generation data sets cannot be PDSEs, UNIX files, or VSAM data sets. The same GDG may contain SMS and non-SMS data sets.

There are usability advantages to grouping related data sets using a function such as GDS. For example, the catalog management routines can refer to the information in a special index called a generation index in the catalog. Thus:

- All of the data sets in the group can be referred to by a common name.
- z/OS is able to keep the generations in *chronological order*.
- Outdated or obsolete generations can be *automatically deleted* from the catalog by z/OS.

Another advantage is the ability to reference a new generation using the same JCL.

GDS's have sequentially ordered *absolute* and *relative* names that represent their age. The catalog management routines use the absolute generation name in the catalog. Older data sets have smaller absolute numbers. The relative name is a signed integer used to refer to the most current (0), the next to most current (-1), and so forth, generation. See also 4.20, "Absolute generation and version numbers" on page 146 and 4.21, "Relative generation numbers" on page 147.

A generation data group (GDG) base is allocated in a catalog before the GDS's are cataloged. Each GDG is represented by a *GDG base* entry. Use the access method services **DEFINE** command to allocate the GDG base (see also 4.19, "Defining a generation data group" on page 144).

The GDG base is a construct that exists only in a user catalog, it does not exist as a data set on any volume. The GDG base is used to maintain the generation data sets (GDS), which are the real data sets.

The number of GDS's in a GDG depends on the limit you specify when you create a new GDG in the catalog.

GDG example

In our example in Figure 4-26, the limit is 5. That means, the GDG can hold a maximum of five GDS's. Our data set name is ABC.GDG. Then, you can access the GDS's by their relative names: ABC.GDG(0) corresponds to the absolute name ABC.GDG.G0005V00. ABC.GDG(-1) would be generation ABC.GDG.G0004V00, and so on. The relative number can also be used to catalog a new generation (+1), which would be generation number 6 with an absolute name of ABC.GDG.G0006V00. Because the limit is 5, the oldest generation (G0001V00) would be rolled-off if you define a new one.

Rolled in and rolled off

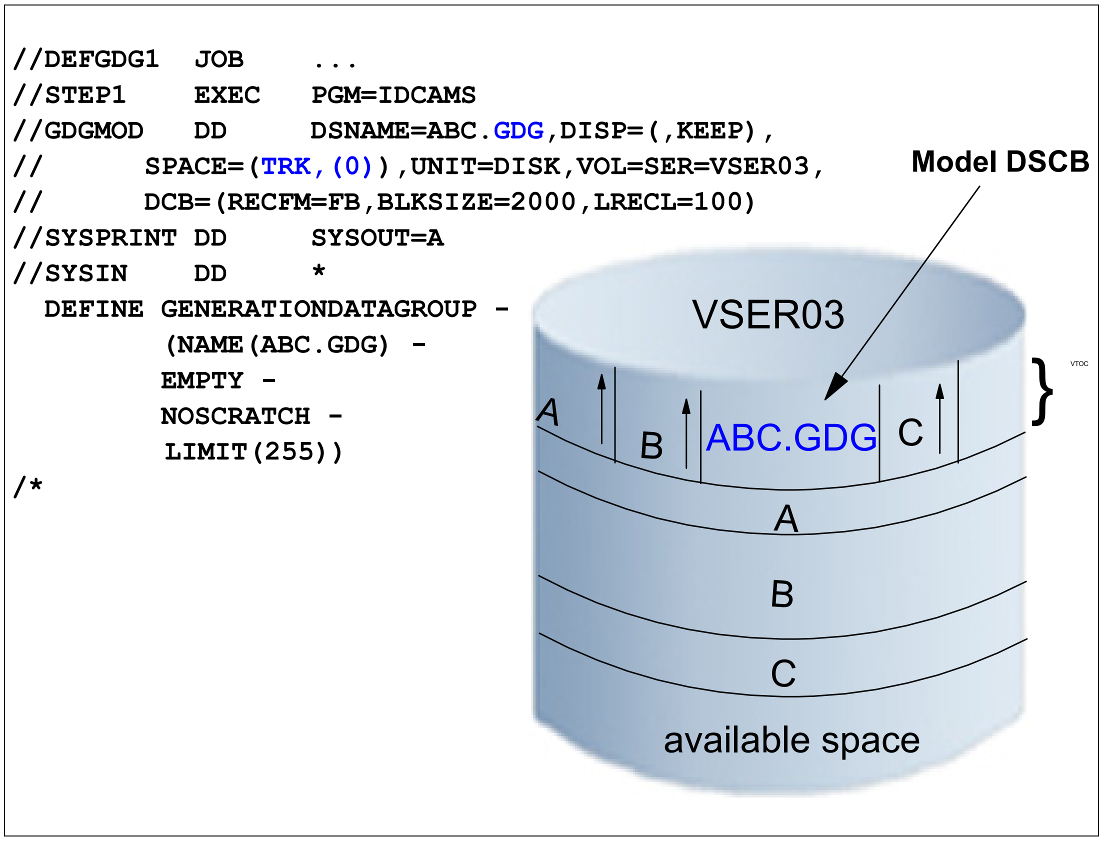
When a GDG contains its maximum number of active generation data sets in the catalog, defined in the **LIMIT** parameter, and a new GDS is rolled in at the end-of-job step, the oldest generation data set is rolled off from the catalog. If a GDG is defined using **DEFINE GENERATIONDATAGROUP EMPTY** and is at its limit, then when a new GDS is rolled in, all the currently active GDS's are rolled off.

The parameters you specify on the **DEFINE GENERATIONDATAGROUP IDCAMS** command determine what happens to rolled-off GDS's. For example, if you specify the **SCRATCH** parameter, the GDS is scratched from VTOC when it is rolled off. If you specify the **NOSCRATCH** parameter, the rolled-off generation data set is re-cataloged as rolled off and is disassociated with its generation data group.

GDS's can be in a deferred roll-in state if the job never reached end-of-step or if they were allocated as DISP=(NEW,KEEP) and the data set is not system-managed. However, GDS's in a deferred roll-in state can be referred to by their absolute generation numbers. You can use the **IDCAMS** command **ALTER ROLLIN** to roll in these GDS's.

For further information about Generation Data Groups, refer to *z/OS DFSMS: Using Data Sets,* SC26-7410.

4.19 Defining a generation data group



```
//DEFGDG1  JOB     ...
//STEP1    EXEC    PGM=IDCAMS
//GDGMOD   DD      DSNAME=ABC.GDG,DISP=(,KEEP),
//      SPACE=(TRK,(0)),UNIT=DISK,VOL=SER=VSER03,
//      DCB=(RECFM=FB,BLKSIZE=2000,LRECL=100)
//SYSPRINT DD      SYSOUT=A
//SYSIN    DD      *
  DEFINE GENERATIONDATAGROUP -
         (NAME(ABC.GDG) -
         EMPTY -
         NOSCRATCH -
         LIMIT(255))
/*
```

Figure 4-27 Defining a GDG

Defining a generation data group

The **DEFINE GENERATIONDATAGROUP** command creates a catalog entry for a generation data group (GDG).

Figure 4-28 shows the JCL to define a GDG.

```
//DEFGDG1  JOB    ...
//STEP1    EXEC   PGM=IDCAMS
//SYSPRINT DD     SYSOUT=A
//SYSIN    DD     *
       DEFINE GENERATIONDATAGROUP -
              (NAME(ABC.GDG) -
              NOEMPTY -
              NOSCRATCH -
              LIMIT(5) )
/*
```

Figure 4-28 JCL to define a GDG catalog entry

The **DEFINE GENERATIONDATAGROUP** command defines a GDG base catalog entry GDG01.

The parameters are:

NAME	Specifies the name of the ABC.GDG. Each GDS in the group will have the name ABC.*GDG.GxxxxVyy*, where *xxxx* is the generation number and *yy* is the version number. See "Absolute generation and version numbers" on page 146.
NOEMPTY	Specifies that only the oldest generation data set is to be uncataloged when the maximum is reached (recommended).
EMPTY	Specifies that all GDS's in the group are to be uncataloged when the group reaches the maximum number of data sets (as specified by the LIMIT parameter) and one more GDS is added to the group.
NOSCRATCH	Specifies that when a data set is uncataloged, its DSCB is not to be removed from its volume's VTOC. Therefore, even if a data set is uncataloged, its records can be accessed when it is allocated to a job step with the appropriate JCL DD statement.
LIMIT	Specifies that the maximum number of GDG data sets in the group is 5. This parameter is required.

Figure 4-29 shows a generation data set defined within the GDG by using JCL statements.

```
//DEFGDG2  JOB    ...
//STEP1    EXEC   PGM=IEFBR14
//GDGDD1   DD     DSNAME=ABC.GDG(+1),DISP=(NEW,CATLG),
//         SPACE=(TRK,(10,5)),VOL=SER=VSER03,
//         UNIT=DISK
//SYSPRINT DD     SYSOUT=A
//SYSIN    DD     *
/*
```

Figure 4-29 JCL to define a generation data set

The job DEFGDG2 allocates space and catalogs a GDG data set in the newly-defined GDG. The job control statement GDGDD1 DD specifies the GDG data set in the GDG.

Creating a model DSCB

As Figure 4-27 shows, you can also create a model DSCB with the same name as the GDG base. You can provide initial DCB attributes when you create your model; however, you need not provide any attributes now. Because only the attributes in the data set label are used, allocate the model data set with SPACE=(TRK,0) to conserve direct access space. You can supply initial or overriding attributes creating and cataloging a generation.

Only one model DSCB is necessary for any number of generations. If you plan to use only one model, do not supply DCB attributes when you create the model. When you subsequently create and catalog a generation, include necessary DCB attributes in the DD statement referring to the generation. In this manner, any number of GDGs can refer to the same model. The catalog and model data set label are always located on a direct access volume, even for a magnetic tape GDG.

Restriction: You cannot use a model DSCB for system-managed generation data sets.

4.20 Absolute generation and version numbers

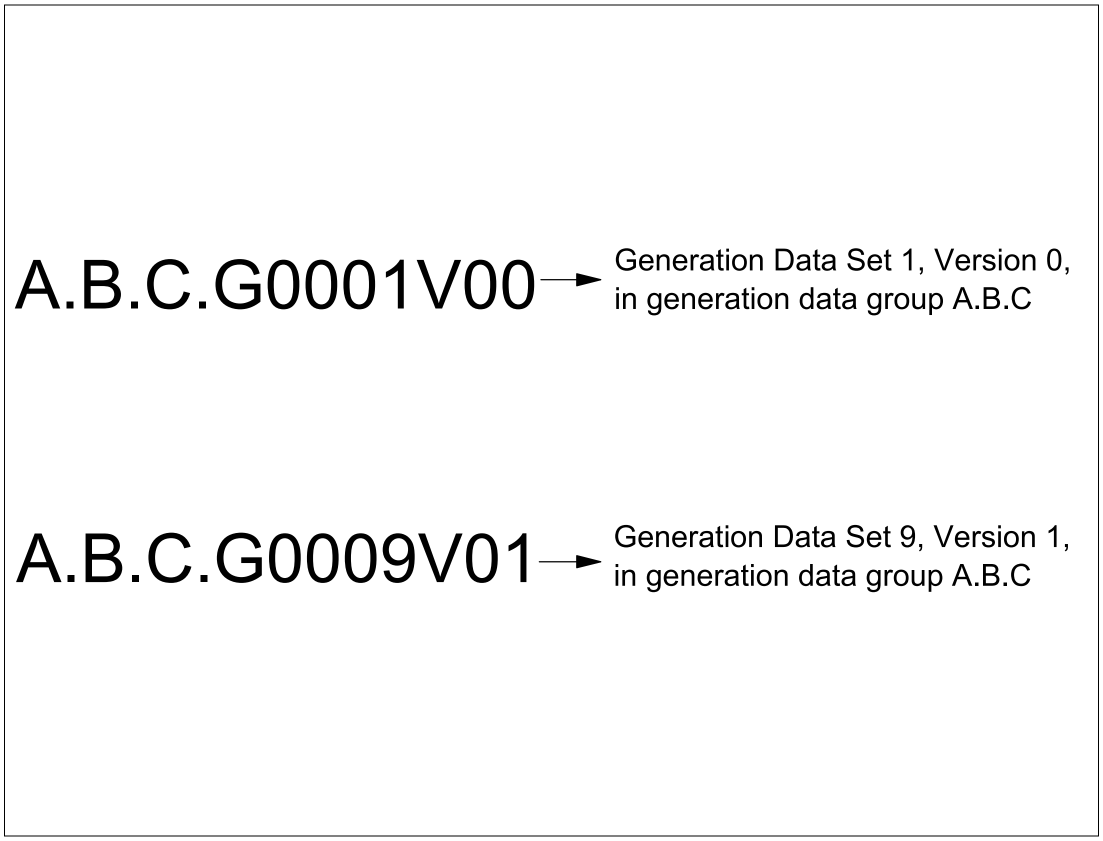

A.B.C.G0001V00
Generation Data Set 1, Version 0, in generation data group A.B.C
A.B.C.G0009V01
Generation Data Set 9, Version 1, in generation data group A.B.C

Figure 4-30 Absolute generation and version numbers

Absolute generation and version numbers

An absolute generation and version number is used to identify a specific generation of a GDG. The same GDS may have different versions, which are maintained by your installation. The version number allows you to perform normal data set operations without disrupting the management of the generation data group. For example, if you want to update the second generation in a three-generation group, replace generation 2, version 0, with generation 2, version 1. Only one version is kept for each generation.

The generation and version number are in the form *GxxxxVyy*, where *xxxx* is an unsigned four-digit decimal generation number (0001 through 9999) and *yy* is an unsigned two-digit decimal version number (00 through 99). For example:

- A.B.C.G0001V00 is generation data set 1, version 0, in generation data group A.B.C.
- A.B.C.G0009V01 is generation data set 9, version 1, in generation data group A.B.C.

The number of generations and versions is limited by the number of digits in the absolute generation name; that is, there can be 9,999 generations. Each generation can have 100 versions. The system automatically maintains the generation number.

You can catalog a generation using either absolute or relative numbers. When a generation is cataloged, a generation and version number is placed as a low-level entry in the generation data group. To catalog a version number other than V00, you must use an absolute generation and version number.

4.21 Relative generation numbers

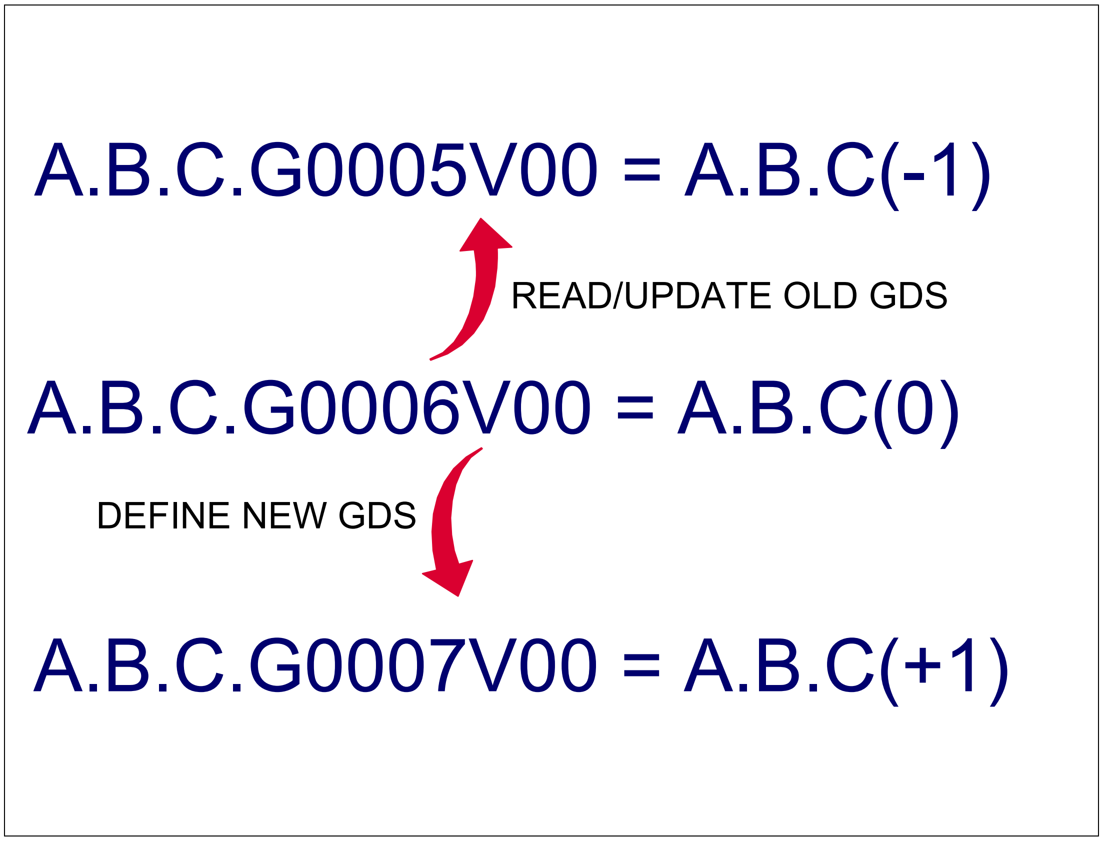

A.B.C.G0005V00 = A.B.C(-1)
READ/UPDATE OLD GDS
A.B.C.G0006V00 = A.B.C(0)
DEFINE NEW GDS
A.B.C.G0007V00 = A.B.C(+1)

Figure 4-31 Relative generation numbers

Relative generation numbers

As an alternative to using absolute generation and version numbers when cataloging or referring to a generation, you can use a relative generation number. To specify a relative number, use the generation data group name followed by a negative integer, a positive integer, or a zero (0), enclosed in parentheses; for example, A.B.C(-1), A.B.C(+1), or A.B.C(0).

The value of the specified integer tells the operating system what generation number to assign to a *new generation data set*, or it tells the system the location of an entry representing a previously cataloged *old generation data set*.

When you use a relative generation number to catalog a generation, the operating system assigns an absolute generation number and a version number of V00 to represent that generation. The absolute generation number assigned depends on the number last assigned and the value of the relative generation number that you are now specifying. For example, if in a previous job generation, A.B.C.G0006V00 was the last generation cataloged, and you specify A.B.C(+1), the generation now cataloged is assigned the number G0007V00.

Though any positive relative generation number can be used, a number greater than 1 can cause absolute generation numbers to be skipped for a new generation data set. For example, if you have a single step job and the generation being cataloged is a +2, one generation number is skipped. However, in a multiple step job, one step might have a +1 and a second step a +2, in which case no numbers are skipped. The mapping between relative and absolute numbers is kept until the end of the job.

4.22 Partitioned organized (PO) data sets

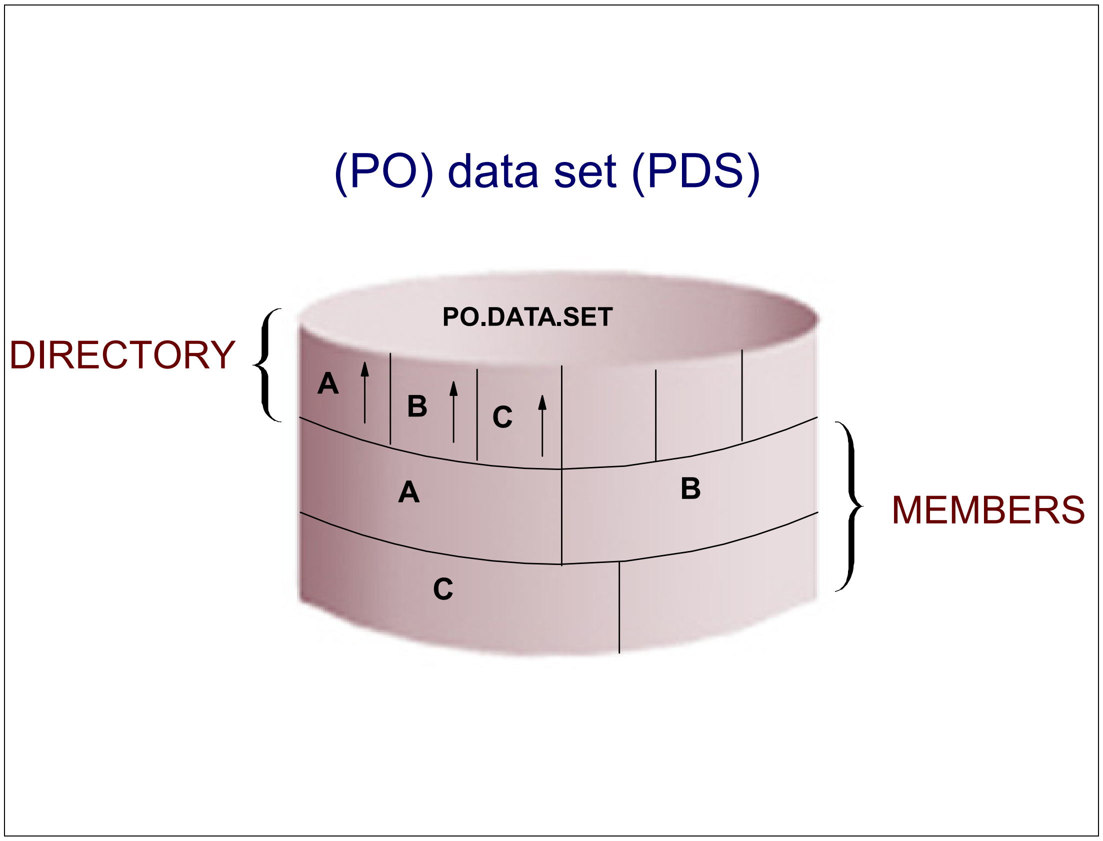


Figure 4-32 Partitioned organized (PO) data set

Partitioned organized (PO) data sets

Partitioned data sets are similar in organization to a library and are often referred to this way. Normally, a library contains a great number of “books,” and sorted directory entries are used to locate them.

In a partitioned organized data set, the “books” are called *members,* and to locate them, they are pointed to by entries in a *directory*, as shown in Figure 4-32.

The members are individual sequential data sets and can be read or written sequentially, once they have been located via directory. Then, the records of a given member are written or retrieved sequentially.

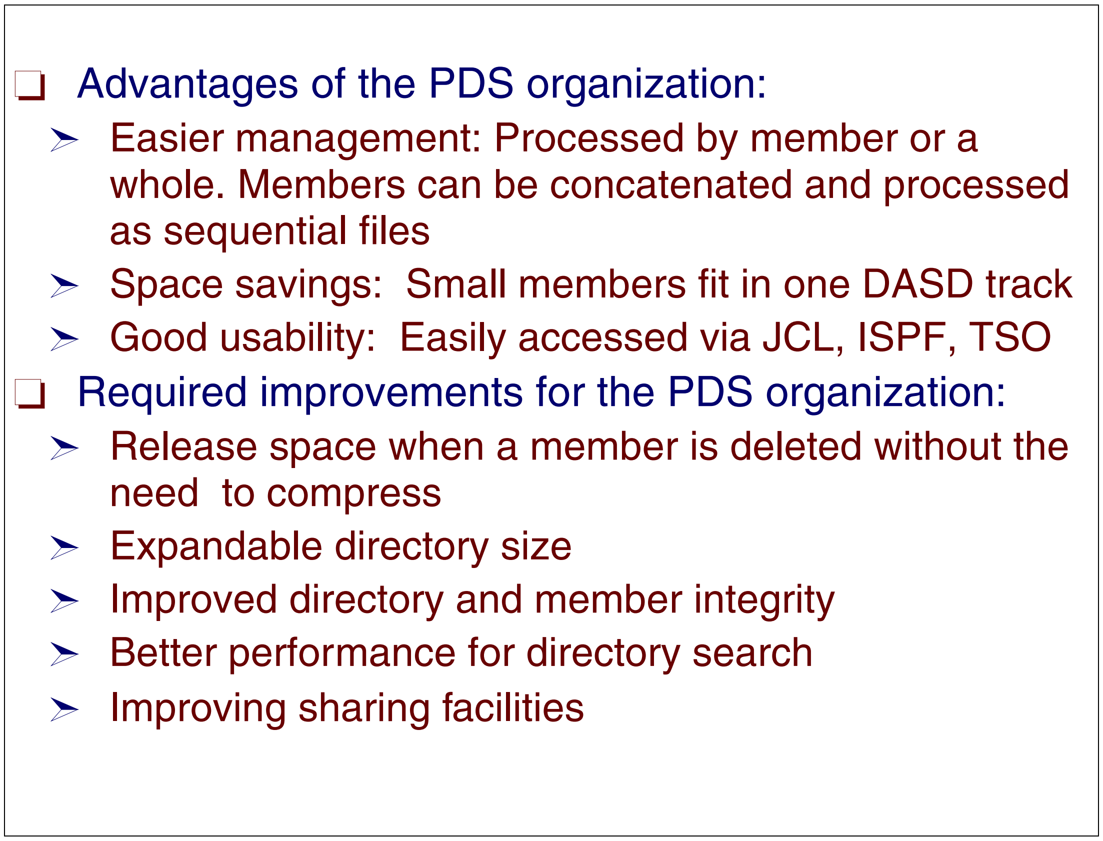
Partitioned data sets can only exist on DASD. Each member has a unique name, one to eight characters in length, and is stored in a directory that is part of the data set.

The main advantage of using a PO data set is that, without searching the entire data set, you can retrieve any individual member after the data set is opened. For example, in a program library (always a partitioned data set) each member is a separate program or subroutine. The individual members can be added or deleted as required.

There are two types of PO data sets:

- Partitioned data set (PDS)
- Partitioned data set extended (PDSE)

4.23 PDS data set organization



- Advantages of the PDS organization:
  - Easier management: Processed by member or a whole. Members can be concatenated and processed as sequential files
  - Space savings: Small members fit in one DASD track
  - Good usability: Easily accessed via JCL, ISPF, TSO
- Required improvements for the PDS organization:
  - Release space when a member is deleted without the need to compress
  - Expandable directory size
  - Improved directory and member integrity
  - Better performance for directory search
  - Improving sharing facilities

Figure 4-33 PDS data organization

Partitioned data set (PDS)

A PDS is stored in only one DASD device. It is divided into sequentially organized members, each described by one or more directory entries. It is an MVS data organization that offers such useful features as:

- Easier management, which makes grouping related data sets under a single name and managing MVS data easier. Files stored as members of a PDS can be processed individually, or all the members can be processed as a unit.
- Space savings so that small members fit in just one DASD track.
- Good usability so that members of a PDS can be used as sequential data sets, and they can be concatenated to sequential data sets. They are also easy to create with JCL, or ISPF, and they are easy to manipulate with ISPF utilities or TSO commands.

However, there are requirements for improvement regarding PDS organization:

- There is no mechanism to reuse the area that contained a deleted or rewritten member. This unused space must be reclaimed by the use of the IEBCOPY utility function called *compression*. See "IEBCOPY utility" on page 118 for information about this utility.
- Directory size is not expandable, causing an overflow exposure. The area for members can grow using secondary allocations. However, this is not true for the directory.
- A PDS has no mechanism to prevent a directory from being overwritten if a program mistakenly opens it for sequential output. If this happens, the directory is destroyed and all the members are lost.

Also, PDS DCB attributes can be easily changed by mistake. If you add a member whose DCB characteristics differ from those of the other members, you change the DCB attributes of the entire PDS, and all the old members become unusable. In this case there is a work around in the form of some tricky code to correct the problem.

- Better directory search time: Entries in the directory are physically ordered by the collating sequence of the names in the members they are pointing to. Any inclusion may cause the full rearrange of the entries.

  There is also no index to the directory entries. The search is sequential using a CKD format. If the directory is big, the I/O operation takes more time. To minimize this, a strong directory buffering has been used (library lookaside (LLA)) for load modules in z/OS.

- Improved sharing facilities: To update or create a member of a PDS, you need exclusive access to the entire data set.

All these improvements require almost total compatibility, at the program level and the user level, with the old PDS.

Allocating space for a PDS

To allocate a PDS, specify PDS in the DSNTYPE parameter and the number of directory blocks in the SPACE parameter, in either the JCL or the SMS data class. You must specify the number of directory blocks, or the allocation fails.

If your data set is large, or if you expect to update it extensively, it might be best to allocate a large space. A PDS cannot occupy more than 65,535 tracks and cannot extend beyond one volume. If your data set is small or is seldom changed, let SMS calculate the space requirements to avoid wasted space or wasted time used for recreating the data set.

Space for the directory is expressed in 256 byte blocks. Each block contains from 3 to 21 entries, depending on the length of the user data field. If you expect 200 directory entries, request at least 10 blocks. Any unused space on the last track of the directory is wasted unless there is enough space left to contain a block of the first member.

The following DD statement defines a new partitioned data set:

```
//DD2  DD  DSNAME=OTTO.PDS12,DISP=(,CATLG),
//         SPACE=(CYL,(5,2,10),,CONTIG)
```

The system allocates 5 cylinders to the data set, of which ten 256-byte records are for a directory. Since the CONTIG subparameter is coded, the system allocates 10 contiguous cylinders on the volume. The secondary allocation is two cylinders, which is needed when the data set needs to expand beyond the five cylinders primary allocation.

4.24 Partitioned data set extended (PDSE)

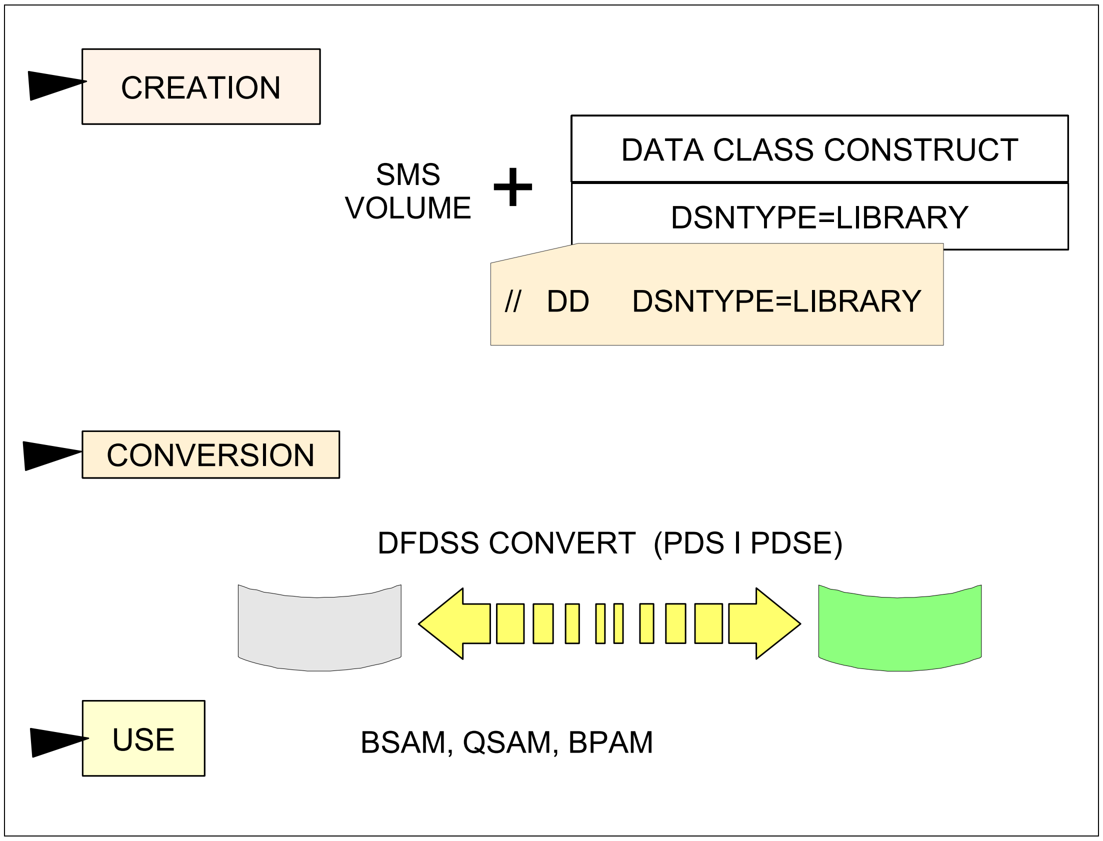


Figure 4-34 PDSE structure

Partitioned data set extended (PDSE)

Partitioned data set extended (PDSE) is a type of data set organization that improves the partition data set (PDS) organization. It has an improved indexed directory structure and a different member format. You can use PDSE for source (programs and text) libraries, macros, and program object (the name of executable code when loaded in PDSE) libraries.

Logically, a PDSE directory is similar to a PDS directory. It consists of a series of directory records in a block. Physically, it is a set of "pages" at the front of the data set, plus additional pages interleaved with member pages. Five directory pages are initially created at the same time as the data set.

New directory pages are added, interleaved with the member pages, as new directory entries are required. A PDSE always occupies at least five pages of storage.

The directory is like a KSDS index structure (KSDS is covered in "VSAM key sequenced cluster (KSDS)" on page 169), making a search much faster. It cannot be overwritten by being opened for sequential output.

If you try to add a member with DCB characteristics that differs from the rest of the members, you will get an error.

There is no longer a need for a PDSE data set to be SMS managed.

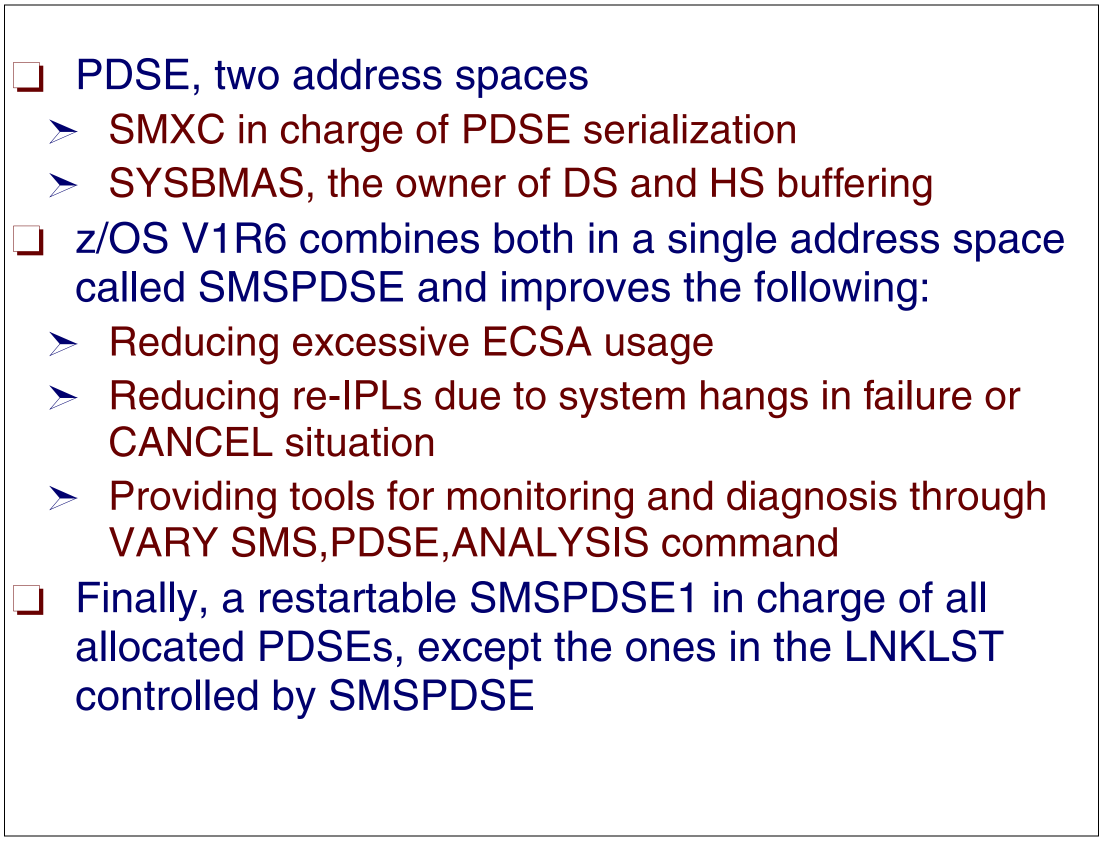
Advantages of PDSEs

PDSE advantages when compared with PDS are:

- The size of a PDSE directory is flexible and can expand to accommodate the number of members stored in it (the size of a PDS directory is fixed at allocation time).
- PDSE members are indexed in the directory by member name. This eliminates the need for time-consuming sequential directory searches holding channels for a long time.
- The logical requirements of the data stored in a PDSE are separated from the physical (storage) requirements of that data, which simplifies data set allocation.
- PDSE automatically reuses space, without the need for an IEBCOPY compress. A list of available space is kept in the directory. When a PDSE member is updated or replaced, it is written in the first available space. This is either at the end of the data set, or in a space in the middle of the data set marked for reuse. For example, by moving or deleting a PDSE member, you free space that is immediately available for the allocation of a new member. This makes PDSEs less susceptible to space-related abends than are PDSs. This space does *not* have to be contiguous. The objective of the space reuse algorithm is to avoid extending the data set unnecessarily.
- The number of PDSE members stored in the library can be large or small without concern for performance or space considerations.
- You can open a PDSE member for output or update without locking the entire data set. The sharing control is at the member level, not the data set level.
- The ability to update a member in place is possible with PDSs and PDSEs. But with PDSEs, you can extend the size of members and the integrity of the library is maintained while simultaneous changes are made to separate members within the library.
- The maximum number of extents of a PDSE is 123; the PDS is limited to 16.
- PDSEs are device-independent because they do not contain information that depends on location or device geometry.
- All members of a PDSE are re-blockable.
- PDSEs can contain program objects built by the program management binder that cannot be stored in PDSs.
- You can share PDSEs within and across systems using PDSESHARING(EXTENDED) in the IGDSMSxx member in the SYS1.PARMLIB. Multiple users are allowed to read PDSE members while the data set is open. You can extend the sharing to enable multiple users on multiple systems to concurrently create new PDSE members and read existing members. (see also “SMS PDSE support” on page 282).
- For performance reasons, directories can be buffered in dataspaces and members can be buffered in hiperspaces by using the MSR parameter in the SMS storage class.
- Replacing a member without replacing all of its aliases deletes all aliases.
- An unlimited number of tracks per volume are now available for PDSEs, that is, more than the previous limit of 65535.

Restriction: You cannot use a PDSE for certain system data sets that are opened in the IPL/NIP time frame.

4.25 PDSE enhancements



- ❑ PDSE, two address spaces
  - ➢ SMXC in charge of PDSE serialization
  - ➢ SYSBMAS, the owner of DS and HS buffering
- ❑ z/OS V1R6 combines both in a single address space called SMSPDSE and improves the following:
  - ➢ Reducing excessive ECSA usage
  - ➢ Reducing re-IPLs due to system hangs in failure or CANCEL situation
  - ➢ Providing tools for monitoring and diagnosis through VARY SMS,PDSE,ANALYSIS command
- ❑ Finally, a restartable SMSPDSE1 in charge of all allocated PDSEs, except the ones in the LNKLST controlled by SMSPDSE

Figure 4-35 PDSE enhancements

PDSE enhancements

Recent enhancements have made PDSEs more reliable and available, correcting a few problems that caused some IPLs due to a hang, deadlock, or out-of-storage condition.

Originally, in order to implement PDSE, two system address spaces were introduced:

- SMXC, in charge of PDSE serialization.
- SYSBMAS, the owner of the data space and hiperspace buffering.

z/OS V1R6 combines SMXC and SYSBMAS to a single address space called SMSPDSE. This improves overall PDSE usability and reliability by:

- Reducing excessive ECSA usage (by moving control blocks into the SMSPDSE address space)
- Reducing re-IPLs due to system hangs in failure or CANCEL situations
- Providing storage administrators with tools for monitoring and diagnosis through VARY SMS,PDSE,ANALYSIS command (for example, determining which systems are using a particular PDSE)

However, the SMSPDSE address space is usually non-restartable because of the eventual existence of perennial PDSEs data sets in the LNKLST concatenation. Then, any hang condition could cause an unplanned IPL. To fix this, we have a new AS, the restartable SMSPDSE1, which is in charge of all allocated PDSEs except the ones in the LNKST.

SMSPDSE1 is created if the following parameters are defined in IGDSMSxx:

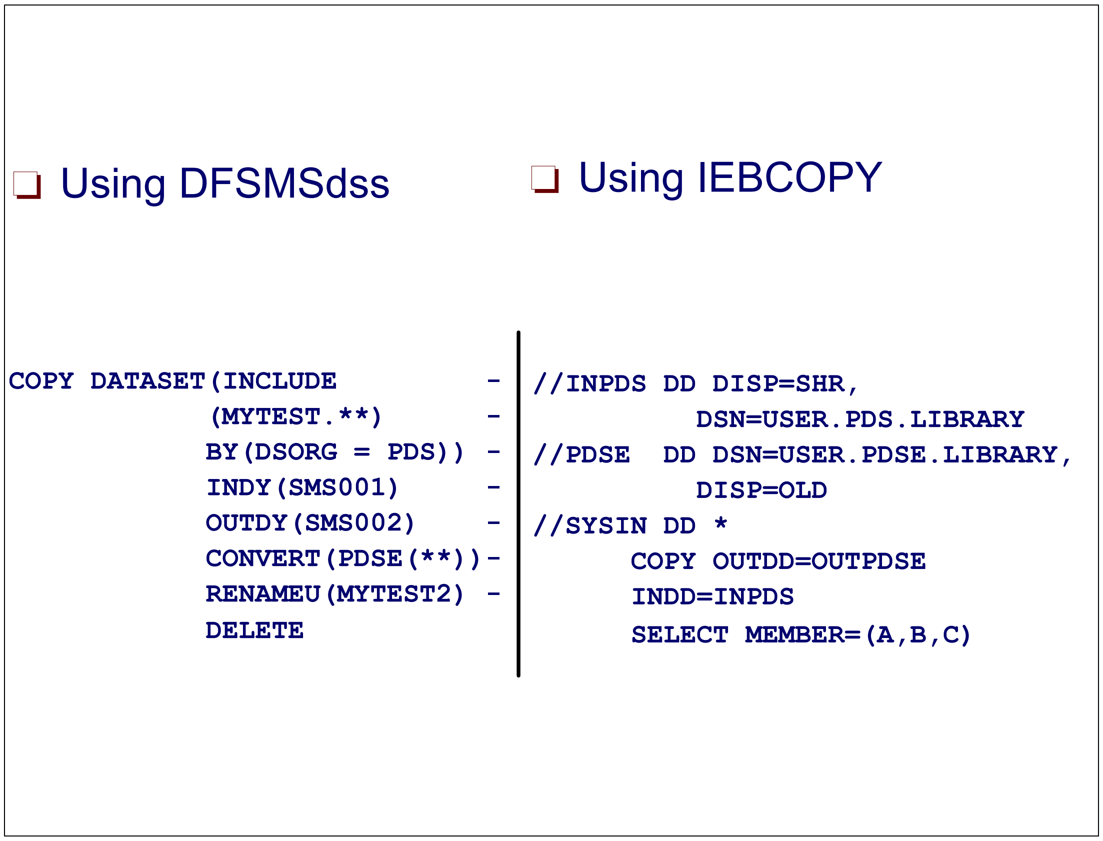
- PDSESHARING(EXTENDED)
- PDSE_RESTARTABLE_AS(YES)

z/OS V1R8 enhancements

With z/OS V1R8, the following improvements for PDSEs are introduced:

- SMSPDSE and SMSPDSE1 support 64-bit addressing, allowing more concurrently opened PDSE members.
- The buffer pool can be retained beyond PDSE close, improving performance due to less buffer pool deletions and creation. This is done by setting PDSE1_BUFFER_BEYOND_CLOSE(YES) in the IGDSMSxx member in parmlib.
- You can dynamically change the amount of virtual storage allocated to hiperspaces for SMSPDSE1 through the SETSMS PDSE1HSP_SIZE(nnn) command.

4.26 PDSE: Conversion



❑ Using DFSMSdss

```
COPY DATASET(INCLUDE            -
            (MYTEST.**)         -
            BY(DSORG = PDS))    -
            INDY(SMS001)        -
            OUTDY(SMS002)       -
            CONVERT(PDSE(**))   -
            RENAMEU(MYTEST2)    -
            DELETE
```

❑ Using IEBCOPY

```
//INPDS DD DISP=SHR,
          DSN=USER.PDS.LIBRARY
//PDSE  DD DSN=USER.PDSE.LIBRARY,
          DISP=OLD
//SYSIN DD *
      COPY OUTDD=OUTPDSE
      INDD=INPDS
      SELECT MEMBER=(A,B,C)
```

Figure 4-36 Converting PDS to PDSE

Converting a PDS data set to a PDSE

You can use **IEBCOPY** or DFSMSdss **COPY** to convert PDS to PDSE, as shown in Figure 4-36. We recommend using DFSMSdss.

You can convert the entire data set or individual members, and also back up and restore PDSEs. By using the DFSMSdss **COPY** function with the CONVERT and PDS keywords, you can convert a PDSE back to a PDS. This is especially useful if you need to prepare a PDSE for migration to a site that does not support PDSEs. When copying members from a PDS load module library into a PDSE program library, or vice versa, the system invokes the program management binder component.

Many types of libraries are candidates for conversion to PDSE, including:

- PDSs that are updated often, and that require frequent and regular reorganization
- Large PDSs that require specific device types because of the size of allocation

Converting PDSs to PDSEs is beneficial, but be aware that certain data sets are unsuitable for conversion to, or allocation as, PDSEs because the system does not retain the original block boundaries.

Using DFSMSdss

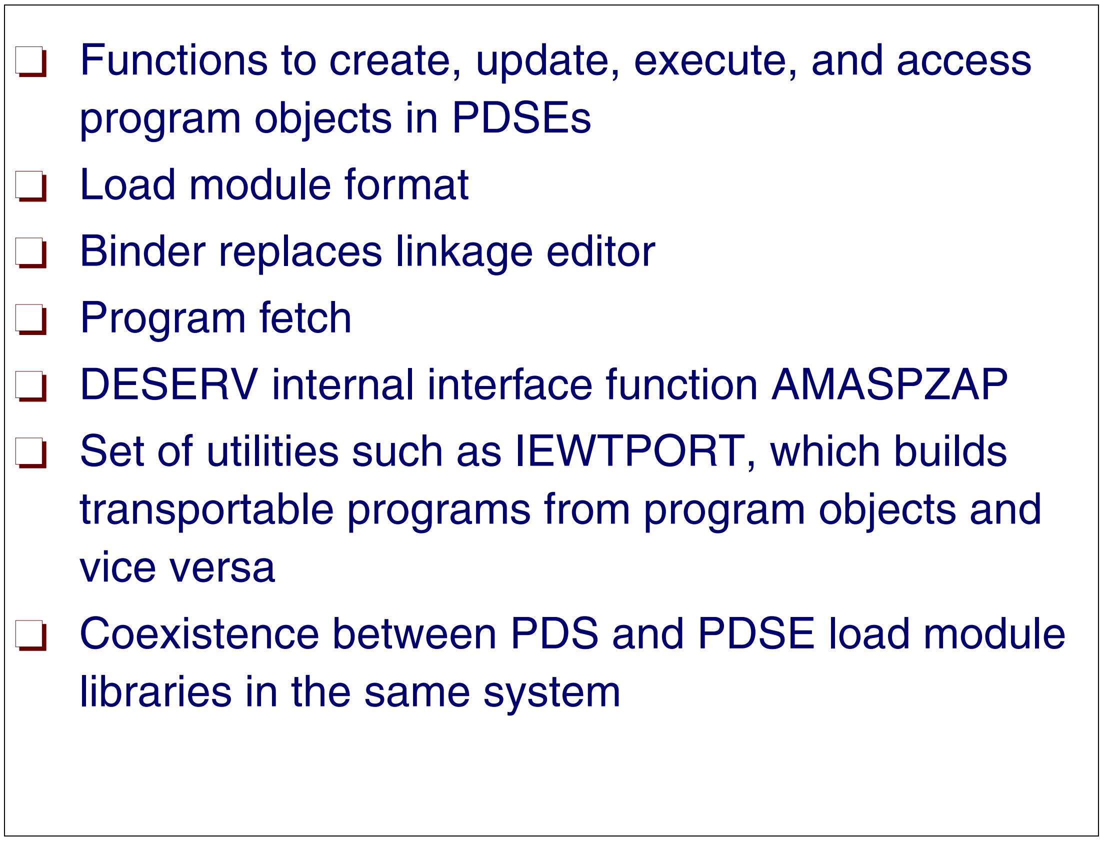
In Figure 4-36, the DFSMSdss COPY example converts all PDSs with the high-level qualifier of “MYTEST” on volume SMS001 to PDSEs with the high-level qualifier of “MYTEST2” on volume SMS002. The original PDSs are then deleted. If you use dynamic allocation, specify INDY and OUTDY for the input and output volumes. However, if you define the ddnames for the volumes, use the INDD and OUDD parameters.

Using IEBCOPY

To copy one or more specific members using IEBCOPY, as shown in Figure 4-36 on page 155, use the SELECT control statement. In this example, IEBCOPY copies members A, B, and C from USER.PDS.LIBRARY to USER.PDSE.LIBRARY.

For more information about DFSMSdss, see *z/OS DFSMSdss Storage Administration Guide,* SC35-0423, and *z/OS DFSMSdss Storage Administration Reference,* SC35-0424.

4.27 Program objects in a PDSE



- Functions to create, update, execute, and access program objects in PDSEs
- Load module format
- Binder replaces linkage editor
- Program fetch
- DESERV internal interface function AMASPZAP
- Set of utilities such as IEWTPORT, which builds transportable programs from program objects and vice versa
- Coexistence between PDS and PDSE load module libraries in the same system

Figure 4-37 Program objects in PDSE

Program objects in PDSE

Program objects are created automatically when load modules are copied into a PDSE. Likewise, program objects are automatically converted back to load modules when they are copied into a partitioned data set. Note that some program objects cannot be converted into load modules because they use features of program objects that do not exist in load modules. A load module is an executable program loaded by the binder in a PDS. A program object is an executable program loaded by the binder in a PDSE.

Load module format

For accessing a PDS directory or member, most PDSE interfaces are indistinguishable from PDS interfaces. However, PDSEs have a different internal format, which gives them increased usability. Each member name can be eight bytes long. The primary name for a program object can be eight bytes long. Alias names for program objects can be up to 1024 bytes long. The records of a given member of a PDSE are written or retrieved sequentially. You can use a PDSE in place of a PDS to store data, or to store programs in the form of program objects. A program object is similar to a load module in a PDS. A load module cannot reside in a PDSE and be used as a load module. One PDSE cannot contain a mixture of program objects and data members.

The binder

The *binder* is the program that processes the output of language translators and compilers into an executable program (load module or program object). It replaced the linkage editor and batch loader. The binder converts the object module output of language translators and compilers into an executable program unit that can either be stored directly into virtual storage for execution or stored in a program library (PDS or PDSE).

MVS program fetch

Most of the loading functions are transparent to the user. When the executable code is in a library, the program management loader component knows whether the program being loaded is a load module or a program object by the source data set type:

- If the program is being loaded from a PDS, it calls IEWFETCH (integrated as part of the loader) to do what it has always done.
- If the program is being loaded from a PDSE, a new routine is called to bring in the program using data-in-virtual (DIV). The loading is done using special loading techniques that can be influenced by externalized options.

The program management loader increases the services of the program fetch component by adding support for loading program objects. The program management loader reads both program objects and load modules into virtual storage and prepares them for execution. It relocates any address constants in the program to point to the appropriate areas in virtual storage and supports 24-bit, 31-bit, and 64-bit addressing ranges. All program objects loaded from a PDSE are page-mapped into virtual storage. When loading program objects from a PDSE, the loader selects a loading mode based on the module characteristics and parameters specified to the binder when you created the program object. You can influence the mode with the binder FETCHOPT parameter. The FETCHOPT parameter allows you to select whether the program is completely preloaded and relocated before execution, or whether pages of the program can be read into virtual storage and relocated only when they are referenced during execution.

DESERV directory service

The directory service DESERV supports both PDS and PDSE libraries. You can issue DESERV in your Assembler program for either PDS or PDSE directory access, but you must pass the DCB address. It does not default to a predefined search order, as does BLDL (the old directory service).

Members of a PDSE program library cannot be rewritten, extended, or updated in place. When updating program objects in a PDSE program library, the AMASPZAP service aid invokes the program management binder, which creates a new version of the program rather than updating the existing version in place.

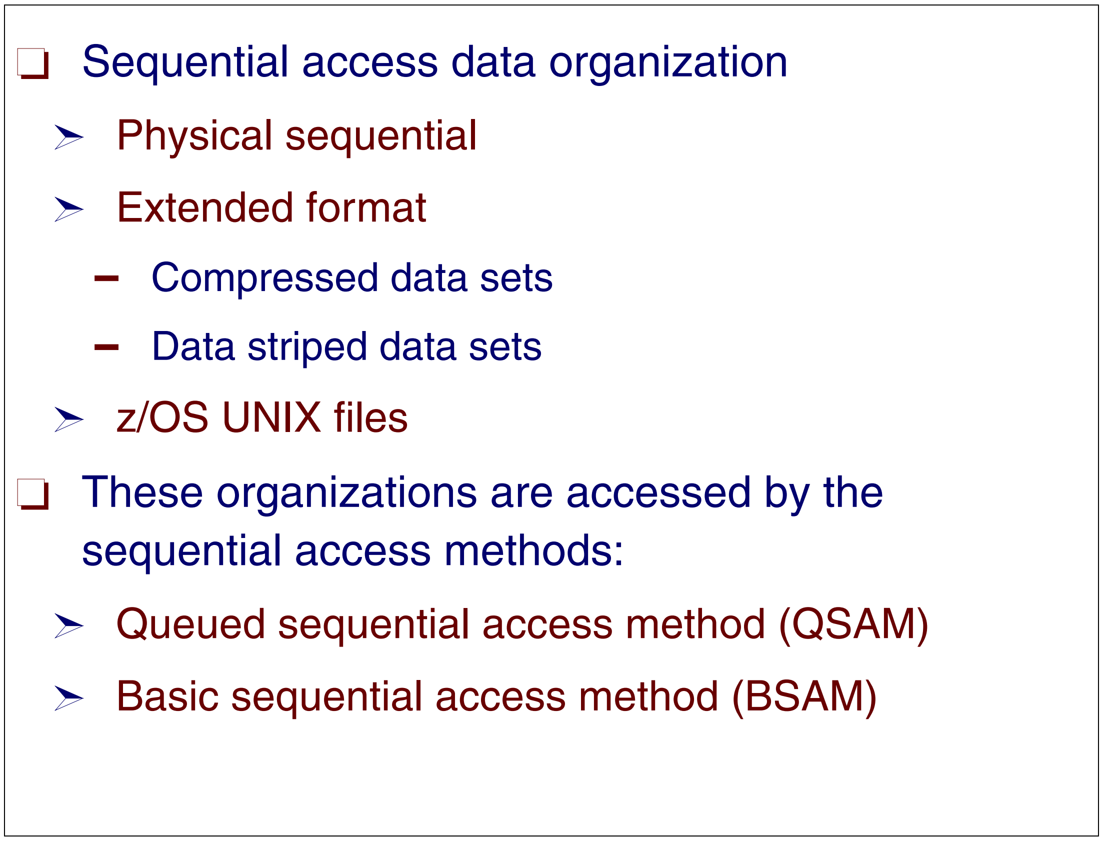
IEWTPORT utility

The transport utility (IEWTPORT) is a program management service with very specific and limited function. It obtains (via the binder) a program object from a PDSE and converts it into a *transportable program file* in a sequential (nonexecutable) format. It also reconstructs the program object from a transportable program file and stores it back into a PDSE (through the binder).

PDS and PDSE coexistence

You can use a PDSE in place of a PDS to store data, or to store programs in the form of program objects. A program object is similar to a load module in a PDS. A load module cannot reside in a PDSE and be used as a load module. One PDSE cannot contain a mixture of program objects and data members. PDSEs and PDSs are processed using the same access methods (BSAM, QSAM, BPAM) and macros, but you cannot use EXCP because of the data set's internal structures.

4.28 Sequential access methods



- ❑ Sequential access data organization
  - ➢ Physical sequential
  - ➢ Extended format
    - – Compressed data sets
    - – Data striped data sets
  - ➢ z/OS UNIX files
- ❑ These organizations are accessed by the sequential access methods:
  - ➢ Queued sequential access method (QSAM)
  - ➢ Basic sequential access method (BSAM)

Figure 4-38 Sequential access methods

Access methods

An access method defines the technique that is used to store and retrieve data. Access methods have their own data set structures to organize data, macros to define and process data sets, and utility programs to process data sets. Access methods are identified primarily by the data set organization. For example, use the basic sequential access method (BSAM) or queued sequential access method (QSAM) with sequential data sets. However, there are times when an access method identified with one organization can be used to process a data set organized in a different manner.

Physical sequential

There are two sequential access methods, basic sequential access method (BSAM) and queued sequential access method (QSAM) and just one sequential organization. Both methods access data organized in a physical sequential manner; the physical records (containing logical records) are stored sequentially in the order in which they are entered.

An important performance item in sequential access is buffering. If you allow enough buffers, QSAM is able to minimize the number of SSCHs by packaging together in the same I/O operation (through CCW command chaining) the data transfer of many physical blocks. This function decreases considerably the total amount of I/O connect time. Another key point is the look-ahead function for reads, that is, reading in advance records that are not yet required by the application program.

Extended format data sets

A special type of this organization is the *extended format data set*. Extended format data sets have a different internal storage format from sequential data sets that are not extended (fixed block with a 32-byte suffix). This storage format gives extended format data sets additional usability and availability characteristics because they:

- Can be allocated in the compressed format (can be referred to as a *compressed format data set*). A compressed format data set is a type of extended format data set that has an internal storage format that allows for data compression.
- Allow data striping, that is, a multivolume sequential file where data can be accessed in parallel.
- Are able to recover from padding error situations.
- Can use the system managed buffering (SMB) technique.

Extended format data sets must be SMS-managed and must reside on DASD. You cannot use an extended format data set for certain system data sets.

z/OS UNIX files

Another type of this organization is the *Hierarchical File System*. HFS files are POSIX-conforming files that reside in an HFS data set. They are byte-oriented rather than record-oriented, as are MVS files. They are identified and accessed by specifying the path leading to them. Programs can access the information in HFS files through z/OS UNIX system calls, such as open(pathname), read(file descriptor), and write(file descriptor).

Programs can also access the information in HFS files through the MVS BSAM, QSAM, and VSAM (Virtual Storage Access Method) access methods. When using BSAM or QSAM, an HFS file is simulated as a multi-volume sequential data set. When using VSAM, an HFS file is simulated as an ESDS. HFS data sets are:

- Supported by standard DADSM create, rename, and scratch
- Supported by DFSMShsm for dump/restore and migrate/recall if DFSMSdss is used as the data mover
- Not supported by IEBCOPY or the DFSMSdss COPY function

QSAM and BSAM

BSAM arranges records sequentially in the order in which they are entered. A data set that has this organization is a sequential data set. The user organizes records with other records into blocks. This is basic access. You can use BSAM with the following data types:

- Basic format sequential data sets, which before z/OS V1R7 were known as sequential data sets, or more accurately as non-extended-format sequential data sets
- Large format sequential data sets
- Extended-format data sets
- z/OS UNIX files

QSAM arranges records sequentially in the order that they are entered to form sequential data sets, which are the same as those data sets that BSAM creates. The system organizes records with other records. QSAM anticipates the need for records based on their order. To improve performance, QSAM reads these records into storage before they are requested. This is called queued access. You can use QSAM with the following data types:

- Sequential data sets
- Basic format sequential data sets before z/OS V1R7, which were known as sequential data sets or more accurately as non-extended-format sequential data sets
- Large format sequential data sets
- Extended-format data sets
- z/OS UNIX files

4.29 Virtual storage access method (VSAM)

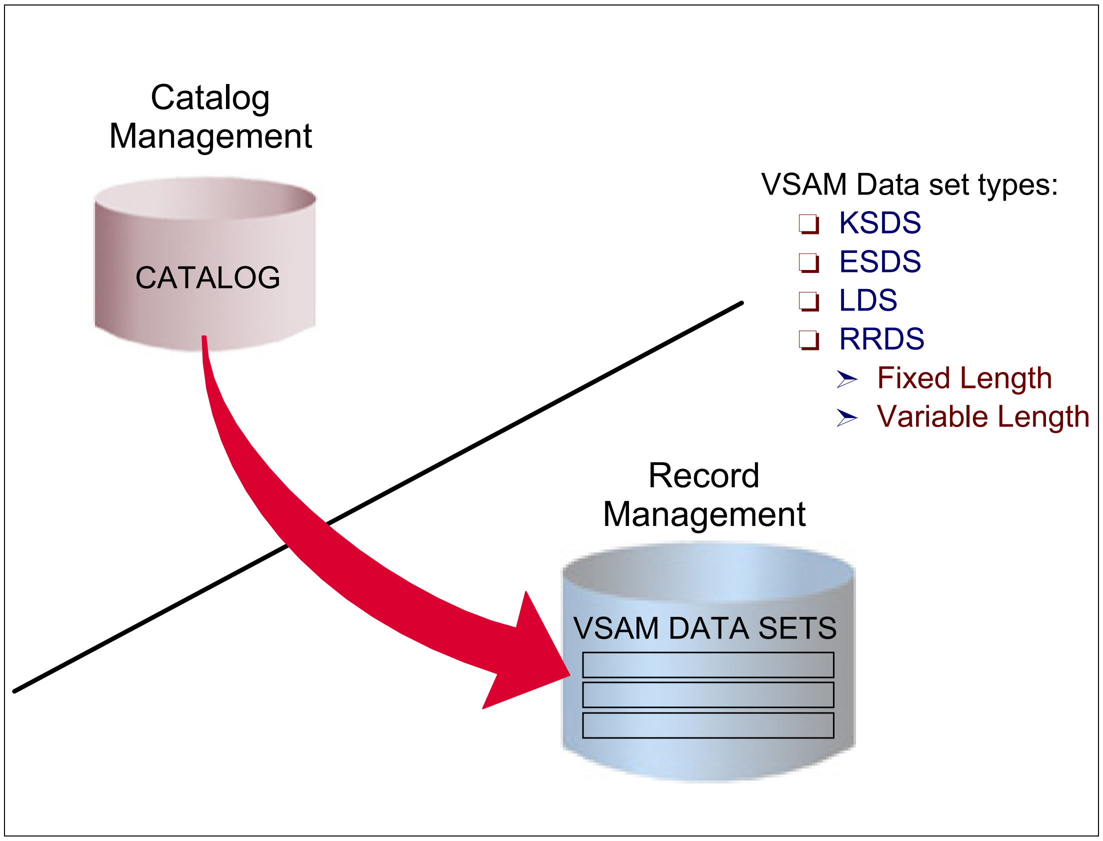


Figure 4-39 Virtual storage access method

Virtual storage access method (VSAM)

VSAM is a DFSMSdfp component used to organize data and maintain information about the data in a catalog. VSAM arranges records by an index key, relative record number, or relative byte addressing. VSAM is used for direct or sequential processing of fixed-length and variable-length records on DASD. Data that is organized by VSAM is cataloged for easy retrieval and is stored in one of five types of data sets.

There are two major parts of VSAM:

- Catalog management - The catalog contains information about the data sets.
- Record management - VSAM can be used to organize records into five types of data sets, that is, the VSAM access method function:
  - Key-sequenced data sets (KSDS) contain records in ascending collating sequence. Records can be accessed by a field, called a key, or by a relative byte address.
  - Entry-sequenced data sets (ESDS) contain records in the order in which they were entered. Records are added to the end of the data set.
  - Linear data sets (LDS) contain data that has no record boundaries. Linear data sets contain none of the control information that other VSAM data sets do. Linear data sets must be cataloged in a catalog.
  - Relative record data sets (RRDS) contains records in relative record number order, and the records can be accessed only by this number. There are two types of relative

record data sets, as follows:

- Fixed-length RRDS: The records must be of fixed length.
- Variable-length RRDS (VRRDS): The records can vary in length.

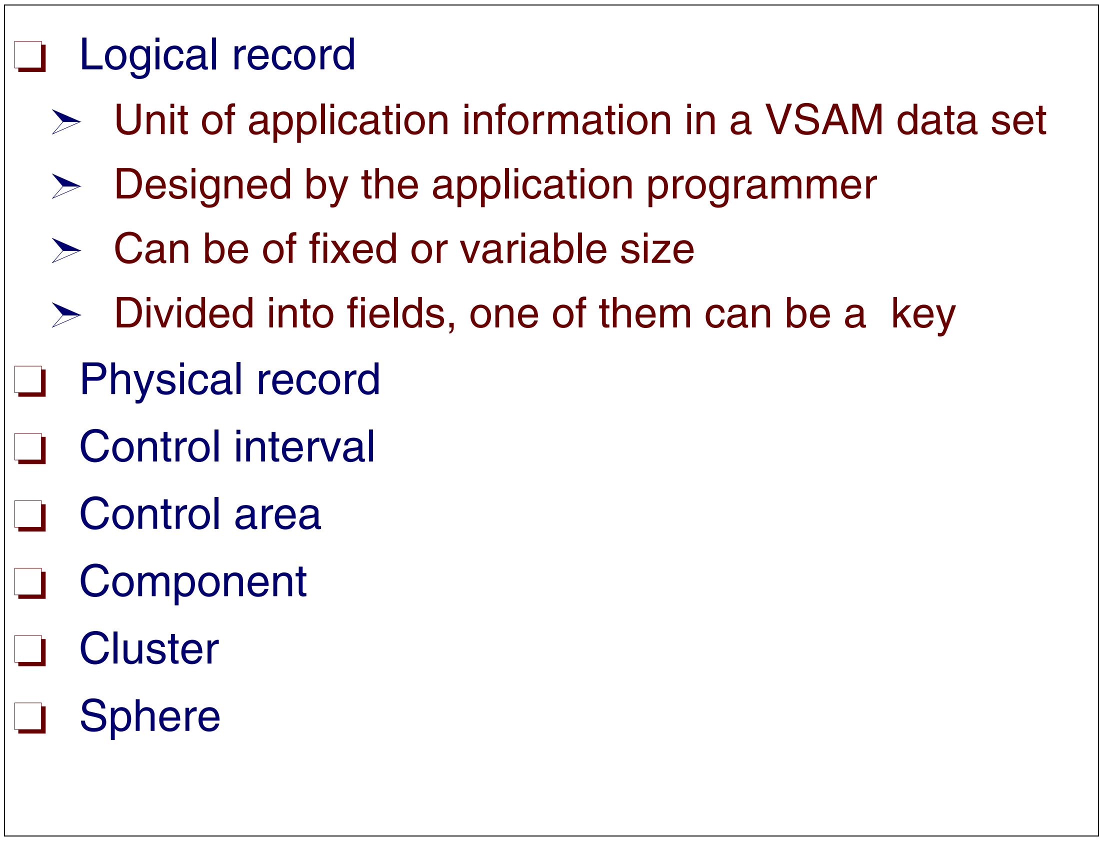
VSAM data sets

The primary difference among these types of data sets is the way in which their records are stored and accessed. VSAM arranges records by an index key, by relative byte address, or by relative record number. Data organized by VSAM must be cataloged and is stored in one of five types of data sets, depending on an application designer option.

z/OS UNIX files can be accessed as though they are VSAM entry-sequenced data sets (ESDS). Although UNIX files are not actually stored as entry-sequenced data sets, the system attempts to simulate the characteristics of such a data set. To identify or access a UNIX file, specify the path that leads to it.

Any type of VSAM data set can be in extended format. Extended-format data sets have a different internal storage format than data sets that are not extended. This storage format gives extended-format data sets additional usability characteristics and possibly better performance due to striping. You can choose that an extended-format key-sequenced data set be in the compressed format. Extended-format data sets must be SMS managed. You cannot use an extended-format data set for certain system data sets.

4.30 VSAM terminology



- Logical record
  - Unit of application information in a VSAM data set
  - Designed by the application programmer
  - Can be of fixed or variable size
  - Divided into fields, one of them can be a key
- Physical record
- Control interval
- Control area
- Component
- Cluster
- Sphere

Figure 4-40 VSAM terminology

Logical record

A *logical record* is a unit of application information used to store data in a VSAM cluster. The logical record is designed by the application programmer from the business model. The application program, through a GET, requests that a specific logical record be moved from the I/O device to memory in order to be processed. Through a PUT, the specific logical record is moved from memory to an I/O device. A logical record can be of a fixed size or a variable size, depending on the business requirements.

The logical record is divided into fields by the application program, such as the name of the item, code, and so on. One or more contiguous fields can be defined as a key field to VSAM, and a specific logical record can be retrieved directly by its key value.

Logical records of VSAM data sets are stored differently from logical records in non-VSAM data sets.

Physical record

A *physical record* is device-dependent and is a set of logical records moved during an I/O operation by just one CCW (Read or Write). VSAM calculates the physical record size in order to optimize the track space (to avoid many gaps) at the time the data set is defined. All physical records in VSAM have the same length. A physical record is also referred to as a *physical block* or simply a *block*. VSAM may have control information along with logical records in a physical record.

Control interval (CI)

VSAM stores records in control intervals. A *control interval* is a continuous area of direct access storage that VSAM uses to store data records and control information that describes the records. Whenever a record is retrieved from direct access storage, the entire control interval containing the record is read into a VSAM I/O buffer in virtual storage. The desired record is transferred from the VSAM buffer to a user-defined buffer or work area.

Control area (CA)

The control intervals in a VSAM data set are grouped together into fixed-length contiguous areas of direct access storage called *control areas*. A VSAM data set is actually composed of one or more control areas. The number of control intervals in a control area is fixed by VSAM. The maximum size of a control area is one cylinder, and the minimum size is one track of DASD storage. When you specify the amount of space to be allocated to a data set, you implicitly define the control area size. Refer to “VSAM: Control interval (CI)” on page 165, for more information.

Component

A *component* in systems with VSAM is a named, cataloged collection of stored records, such as the data component or index component of a key-sequenced file or alternate index. A component is a set of CAs. It is the VSAM terminology for an MVS data set. A component has an entry in the VTOC. An example of a component can be the data set containing only data for a KSDS VSAM organization.

Cluster

A *cluster* is a named structure consisting of a group of related components. VSAM data sets can be defined with either the `DEFINE CLUSTER` command or the `ALLOCATE` command. The cluster is a set of components that have a logical binding between them. For example, a KSDS cluster is composed of the data component and the index component. The concept of cluster was introduced to make the JCL to access VSAM more flexible. If you want to access a KSDS normally, just use the cluster’s name on a DD card. Otherwise, if you want some special processing with just the data, use the data component name on the DD card.

Sphere

A *sphere* is a VSAM cluster and its associated data sets. The cluster is originally defined with the access method services `ALLOCATE` command, the `DEFINE CLUSTER` command, or through JCL. The most common use of the sphere is to open a single cluster. The base of the sphere is the cluster itself.

4.31 VSAM: Control interval (CI)

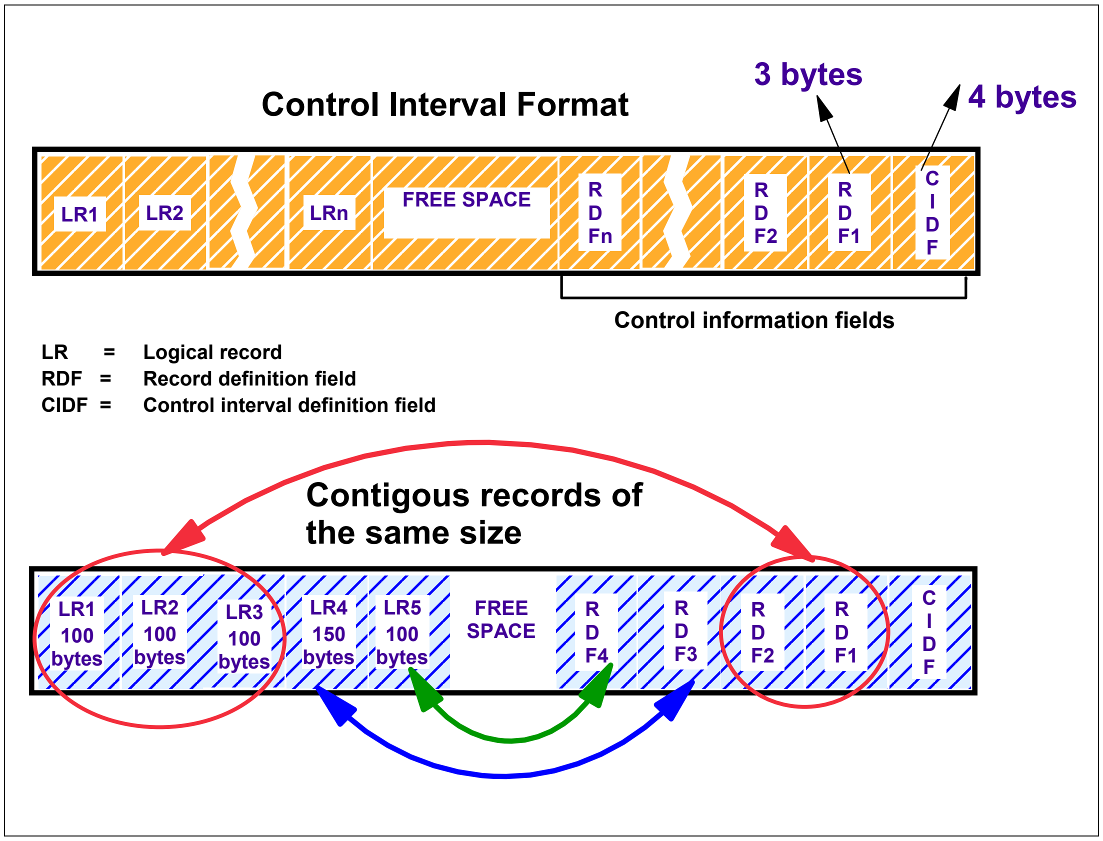


Figure 4-41 Control interval format

Control interval (CI)

The *control interval* is a concept that is unique to VSAM. A CI is formed by one or several physical records (usually just one). It is the fundamental building block of every VSAM file. A CI is a contiguous area of direct access storage that VSAM uses to store data records and control information that describes the records. A CI is the unit of information that VSAM transfers between the storage device and the central storage during one I/O operation. Whenever a logical record is requested by an application program, the entire CI containing the logical record is read into a VSAM I/O buffer in virtual storage. The desired logical record is then transferred from the VSAM buffer to a user-defined buffer or work area (if in move mode).

Based on the CI size, VSAM calculates the best size of the physical block in order to better use the 3390/3380 logical track. The CI size can be from 512 bytes to 32 KB. A CI contents depends on the cluster organization. A KSDS consists of:

- Logical records stored from the beginning to the end of the CI.
- Free space, for data records to be inserted into or lengthened.
- Control information, which is made up of two types of fields:
  - One control interval definition field (CIDF) per CI. CIDF is a 4-byte field. CIDF contains information about the amount and location of free space in the CI.

  – Several record definition fields (RDF) describing the logical records. RDF is a 3-byte field and describes the length of logical records. For fixed length records there are two RDFs, one with the length, and the other with how many records are the same length.

The size of CIs can vary from one component to another, but all the CIs within the data or index component of a particular cluster data set must be of the same length. The CI components and properties may vary, depending on the data set organization. For example, an LDS does not contain CIDFs and RDFs in its CI. All of the bytes in the LDS CI are data bytes.

Spanned records

Spanned records are logical records that are larger than the CI size. They are needed when the application requires very long logical records. To have spanned records, the file must be defined with the SPANNED attribute at the time it is created. Spanned records are allowed to extend across or “span” control interval boundaries, but not beyond control area limits. The RDFs describe whether the record is spanned or not.

A spanned record always begins on a control interval boundary, and fills one or more control intervals within a single control area. A spanned record does *not* share the CI with any other records; in other words, the free space at the end of the last segment is not filled with the next record. This free space is only used to extend the spanned record.

Control area (CA)

Control area is also a concept unique to VSAM. A CA is formed by two or more CIs put together into fixed-length contiguous areas of direct access storage. A VSAM data set is composed of one or more CAs. In most cases, a CA is the size of a 3390/3380 cylinder. The minimum size of a CA is one track. The CA size is implicitly defined when you specify the size of a data set at data set definition.

CAs are needed to implement the concept of splits. The size of a VSAM file is always a multiple of the CA size and VSAM files are extended in units of CAs.

Splits

CI splits and CA splits occur as a result of data record insertions (or increasing the length of an already existing record) in KSDS and VRRDS organizations. If a logical record is to be inserted (in key sequence) and there is not enough free space in the CI, the CI is *split*. Approximately half the records in the CI are transferred to a free CI provided in the CA, and the record to be inserted is placed in the original CI.

If there are no free CIs in the CA and a record is to be inserted, a *CA split* occurs. Half the CIs are sent to the first available CA at the end of the data component. This movement creates free CIs in the original CA, then the record to be inserted causes a CI split.

4.32 VSAM data set components

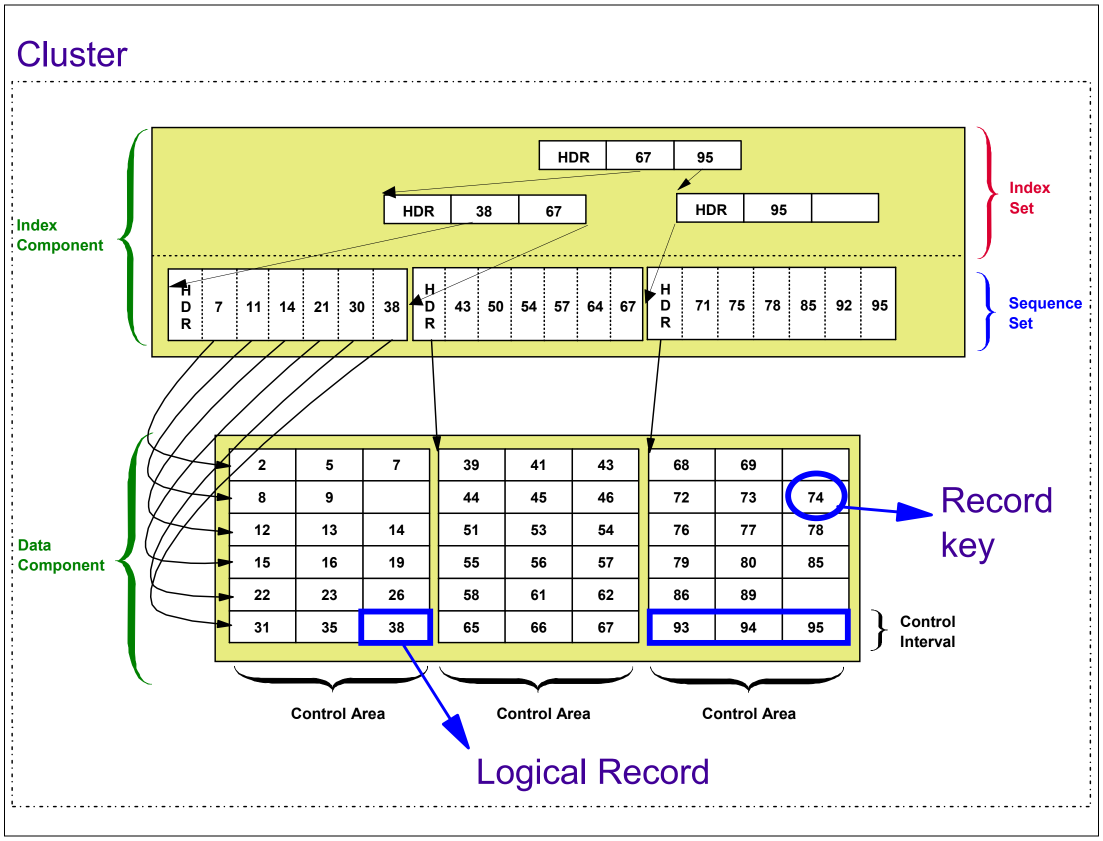


Figure 4-42 VSAM data set components

VSAM data set components

A *component* is an individual part of a VSAM cluster. Each component has a name, an entry in the catalog, and an entry in the VTOC. There are two types of components, the data component and the index component. Some VSAM organizations (such as an ESDS, RRDS, and LDS) have only the data component.

Data component

The *data component* is the part of a VSAM cluster, alternate index, or catalog that contains the data records. All VSAM cluster organizations have the data component.

Index component

The *index component* is a collection of records containing data keys and pointers (relative byte address, or RBA). The data keys are taken from a fixed defined field in each data logical record. The keys in the index logical records are compressed (rear and front). The RBA pointers are compacted. Only KSDS and VRRDS VSAM data set organizations have the index component.

Using the index, VSAM is able to retrieve a logical record from the data component when a request is made randomly for a record with a certain key. A VSAM index can consist of more than one level (binary tree). Each level contains pointers to the next lower level. Because there are random and sequential types of access, VSAM divides the index component into two parts: the sequence set, and the index set.

Sequence set

The *sequence set* is the lowest level of index, and it directly points (through an RBA) to the data CI in the CA of the data component. Each index logical record:

- Occupies one index CI.
- Maps one CI in the data component.
- Contains pointers and high key information for each data CI.
- Contains horizontal pointers from one sequence set CI to the next keyed sequence set CI. These horizontal pointers are needed because of the possibility of splits, which make the physical sequence different from the logical collating sequence by key.

Index set

The records in all levels of the index above the sequence set are called the *index set.* An entry in an index set logical record consists of the highest possible key in an index record in the next lower level, and a pointer to the beginning of that index record. The highest level of the index always contains a single index CI.

The structure of VSAM prime indexes is built to create a single index record at the lowest level of the index. If there is more than one sequence-set-level record, VSAM automatically builds another index level.

Cluster

A *cluster* is the combination of the data component (data set) and the index component (data set) for a KSDS. The cluster provides a way to treat index and data components as a single component with its own name. Use of the word *cluster* instead of *data set* is recommended.

Alternate index (AIX)

The alternate index is a VSAM function that allows logical records of a KSDS or ESDS to be accessed sequentially and directly by more than one key field. The cluster that has the data is called the *base cluster*, then an AIX cluster is built from the base cluster. Alternate indexes eliminate the need to store the same data in different sequences in multiple data sets for the purposes of various applications. Each alternate index is a KSDS cluster consisting of an index component and a data component.

The records in the AIX index component contain the alternate key and the RBA pointing to the alternate index data component. The records in the AIX data component contain the alternate key value itself and all the primary keys corresponding to the alternate key value (pointers to data in the base cluster). The primary keys in the logical record are in ascending sequence within an alternate index value.

Any field in the base cluster record can be used as an alternate key. It can also overlap the primary key (in a KSDS), or any other alternate key. The same base cluster may have several alternate indexes varying the alternate key. There may be more than one primary key value per the same alternate key value. For example, the primary key might be an employee number and the alternate key might be the department name; obviously, the same department name may have several employee numbers.

AIX cluster is created with IDCAMS **`DEFINE ALTERNATEINDEX`** command, then it is populated via the **`BLDINDEX`** command. Before a base cluster can be accessed through an alternate index, a path must be defined. A *path* provides a way to gain access to the base data through a specific alternate index. To define a path, use the **`DEFINE PATH`** command. The utility to issue this command is discussed in “Access method services (IDCAMS)” on page 135.

Sphere

A *sphere* is a VSAM cluster and its AIX associated clusters’ data sets.

4.33 VSAM key sequenced cluster (KSDS)

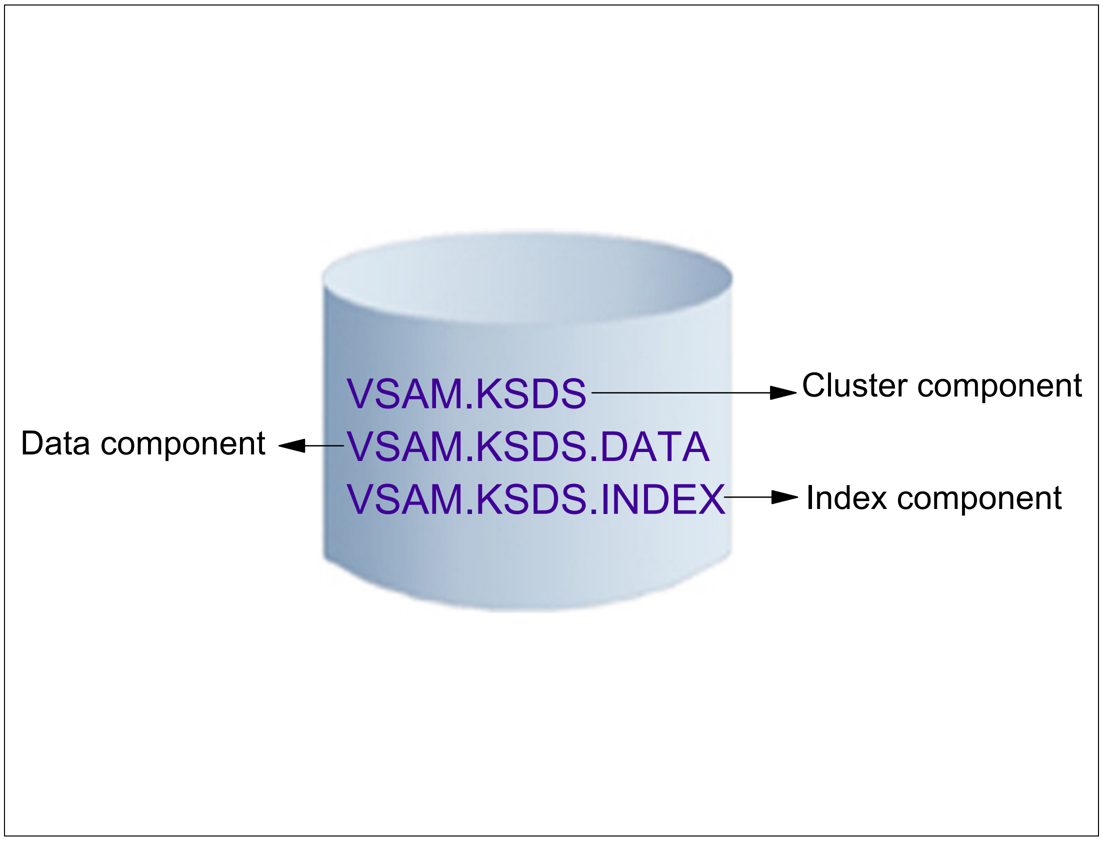


Figure 4-43 Key sequenced cluster (KSDS)

VSAM KSDS cluster

In a KSDS, logical records are placed in the data set in ascending collating sequence by key. The key contains a unique value, which determines the record's collating position in the cluster. The key must be in the same position (off set) in each record.

The key field must be contiguous and each key's contents must be unique. After it is specified, the value of the key cannot be altered, but the entire record may be deleted.

When a new record is added to the data set, it is inserted in its logical collating sequence by key.

A KSDS has a data component and an index component. The index component keeps track of the used keys and is used by VSAM to retrieve a record from the data component quickly when a request is made for a record with a certain key.

A KSDS can have fixed or variable length records.

A KSDS can be accessed in sequential mode, direct mode, or skip sequential mode (meaning that you process sequentially, but directly skip some portions of the data set).

4.34 VSAM: Processing a KSDS cluster

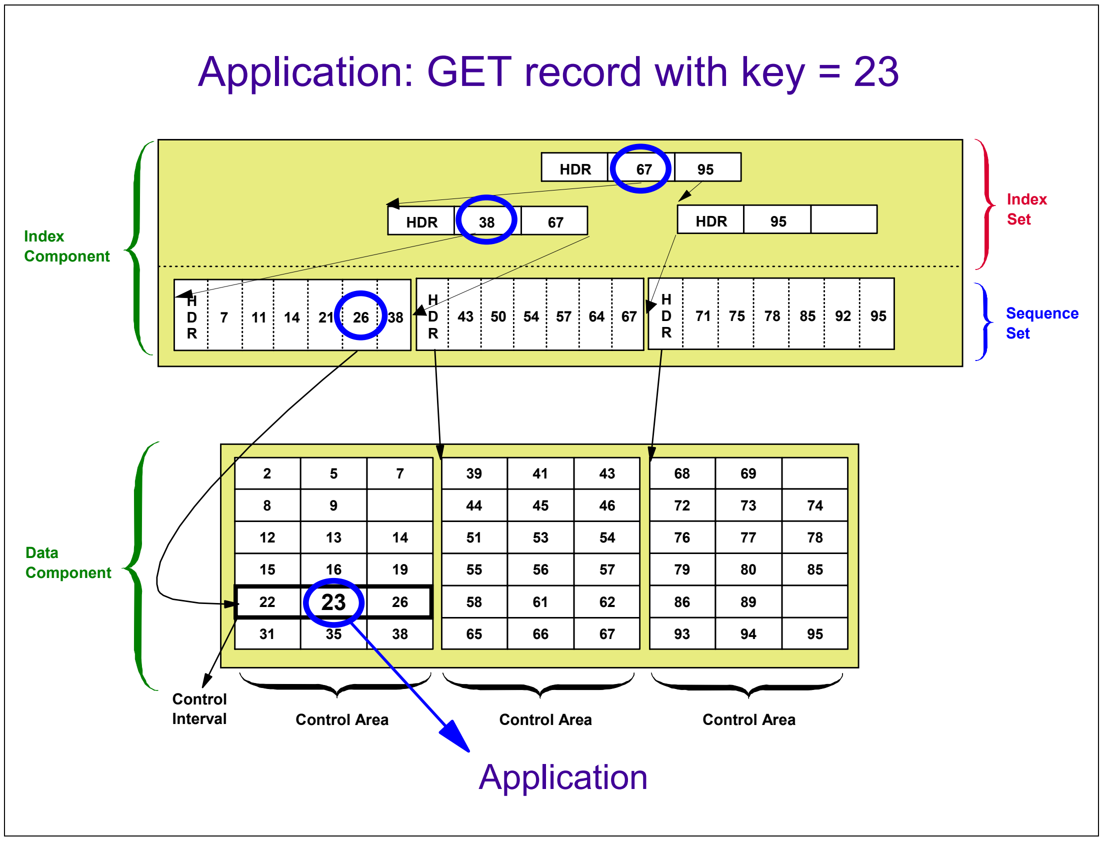


Figure 4-44 Processing an indexed VSAM cluster: Direct access

Processing a KSDS cluster

A KSDS has an index that relates key values to the relative locations in the data set. This index is called the *prime index*. It has two uses:

- Locate the collating position when inserting records
- Locate records for retrieval

When initially loading a KSDS data set, records must be presented to VSAM in key sequence. This loading can be done through the IDCAMS VSAM utility named REPRO. The index for a key-sequenced data set is built automatically by VSAM as the data set is loaded with records.

When a data CI is completely loaded with logical records, free space, and control information, VSAM makes an entry in the index. The entry consists of the highest possible key in the data control interval and a pointer to the beginning of that control interval.

When accessing records sequentially, VSAM refers only to the sequence set. It uses a horizontal pointer to get from one sequence set record to the next record in collating sequence.

Request for data direct access

When accessing records directly, VSAM follows vertical pointers from the highest level of the index down to the sequence set to find vertical pointers to the requested logical record.

Figure 4-44 shows how VSAM searches the index when an application issues a GET for a logical record with key value 23.

The sequence is as follows:

1. VSAM scans the index record in the highest level of the index set for a key that is greater or equal to 23.
2. The entry 67 points to an index record in the next lower level. In this index record, VSAM scans for an entry for a key that is higher than or equal to 23.
3. The entry 38 points to the sequence set that maps the CA holding the CI containing the logical record.
4. VSAM scans the sequence set record with the highest key, searching for a key that is greater than or equal to 23.
5. The entry 26 points to the data component CI that holds the desired record.
6. VSAM searches the CI for the record with key 23. VSAM finds the logical record and gives it to the application program.

If VSAM does not find a record with the desired key, the application receives a return code indicating that the record was not found.

4.35 VSAM entry sequenced data set (ESDS)

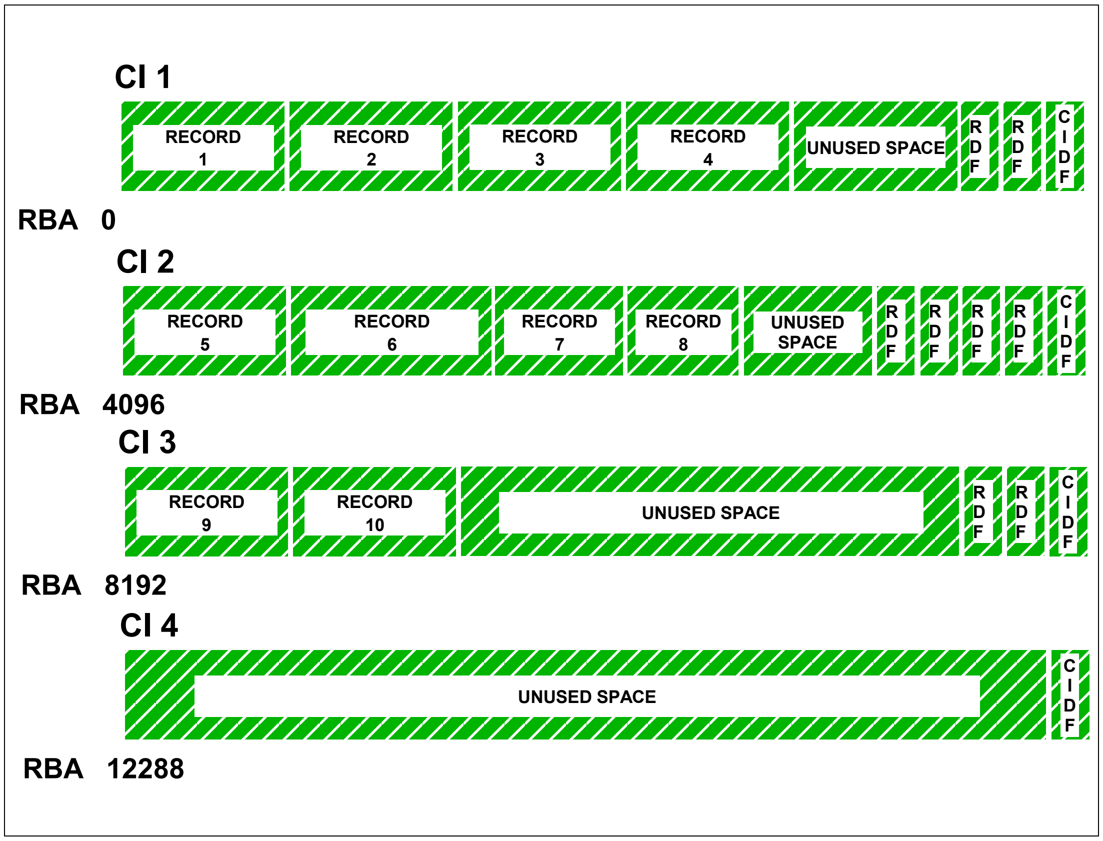


Figure 4-45 VSAM entry sequenced data set (ESDS)

VSAM entry sequenced data set (ESDS)

An *ESDS* is comparable to a sequential data set. It contains fixed-length or variable-length records. Records are sequenced by the order of their entry in the data set, rather than by a key field in the logical record. All new records are placed at the end of the data set. An ESDS cluster has only a data component.

Records can be accessed sequentially or directly by relative byte address (RBA). When a record is loaded or added, VSAM indicates its relative byte address (RBA). The RBA is the offset of the first byte of the logical record from the beginning of the data set. The first record in a data set has an RBA of 0; the second record has an RBA equal to the length of the first record, and so on. The RBA of a logical record depends only on the record's position in the sequence of records. The RBA is always expressed as a full-word binary integer.

Although an entry-sequenced data set does not contain an index component, alternate indexes are allowed. You can build an alternate index with keys to keep track of these RBAs.

4.36 VSAM: Typical ESDS processing

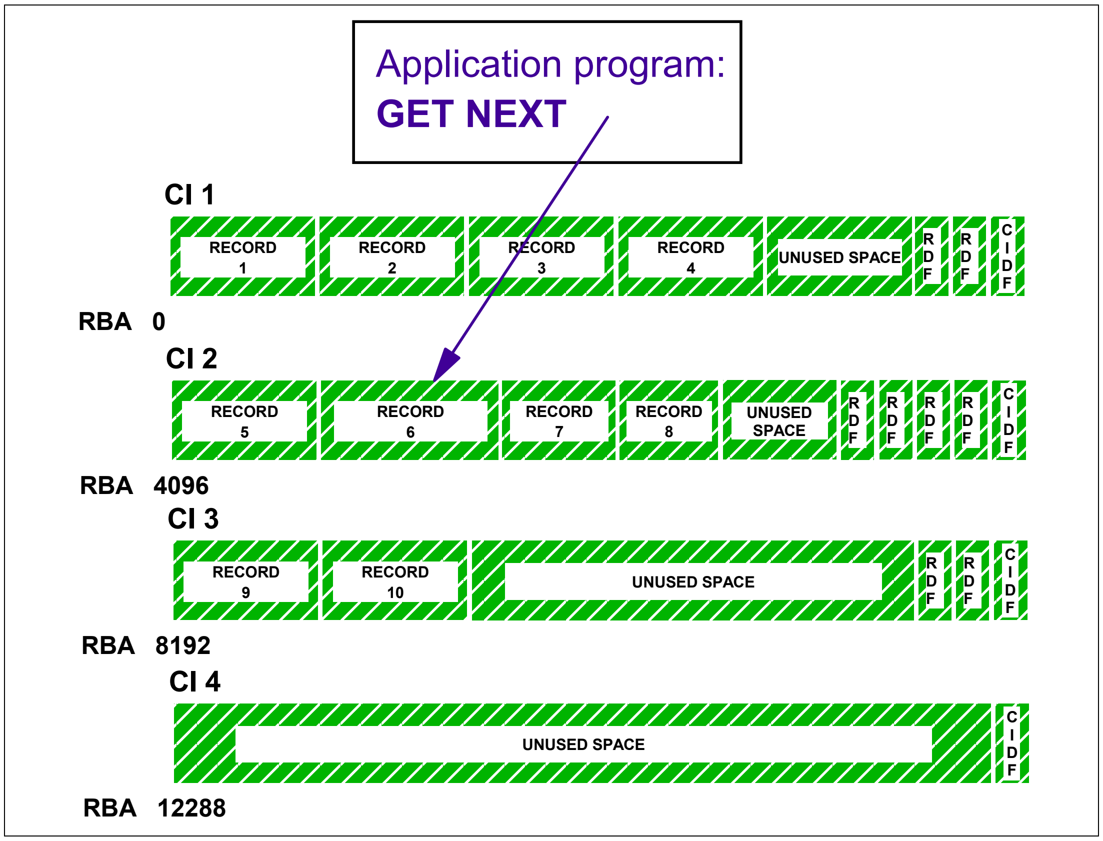


Figure 4-46 Typical ESDS processing (ESDS)

Typical ESDS processing

For an ESDS, two types of processing are supported:

- Sequential access (the most common).
- Direct (or random) access requires the program to give the RBA of the record.

Skip sequential is not allowed.

Existing records can never be deleted. If the application wants to delete a record, it must flag that record as inactive. As far as VSAM is concerned, the record is not deleted. Records can be updated, but without length change.

ESDS organization is suited for sequential processing with variable records, but in a few read accesses you need a direct (random) access by key (here using the AIX cluster).

4.37 VSAM relative record data set (RRDS)

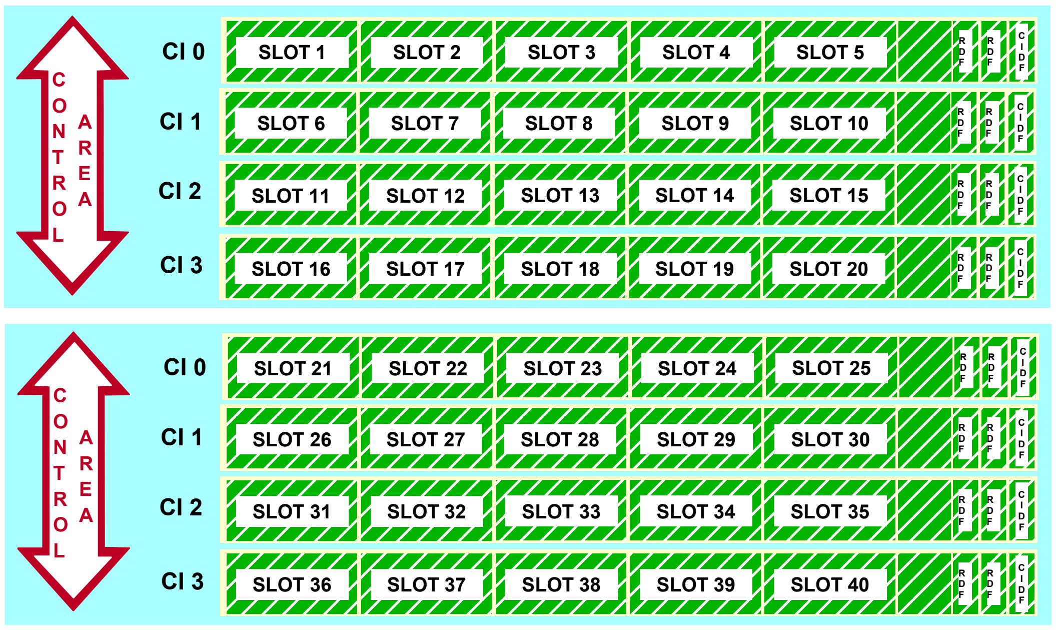


Figure 4-47 VSAM relative record data set (RRDS)

Relative record data set

A relative record data set (RRDS) consists of a number of preformed, fixed length slots. Each slot has a unique relative record number, and the slots are sequenced by ascending relative record number. Each (fixed length) record occupies a slot, and it is stored and retrieved by the relative record number of that slot. The position of a data record is fixed; its relative record number cannot change.

An RRDS cluster has a data component only.

Random load of an RRDS requires a user program implementing such logic.

4.38 VSAM: Typical RRDS processing

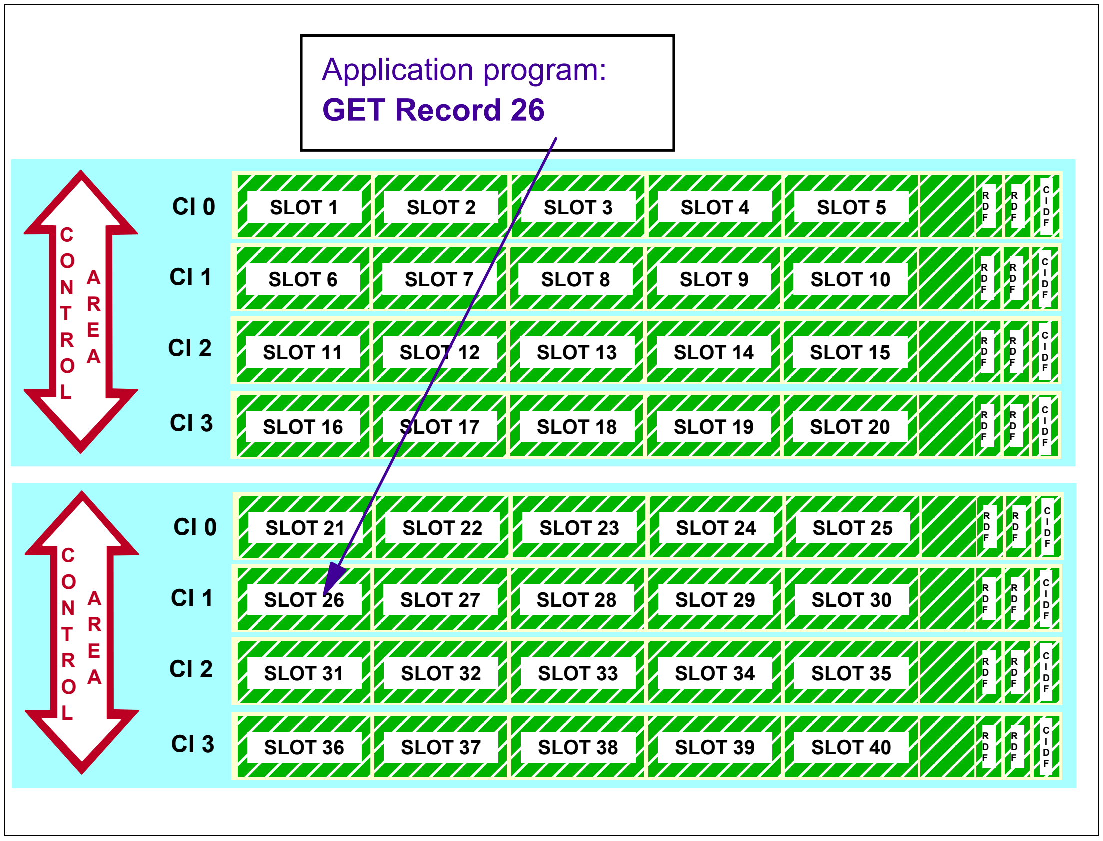


Figure 4-48 Typical RRDS processing

Processing RRDS data sets

The application program inputs the relative record number of the target record. VSAM is able to find its location very quickly by using a formula that takes into consideration the geometry of the DASD device. The relative number is always used as a search argument. For an RRDS, three types of processing are supported:

- Sequential processing.
- Skip-sequential processing.
- Direct processing; in this case, the randomization routine is supported by the application program.

4.39 VSAM linear data set (LDS)

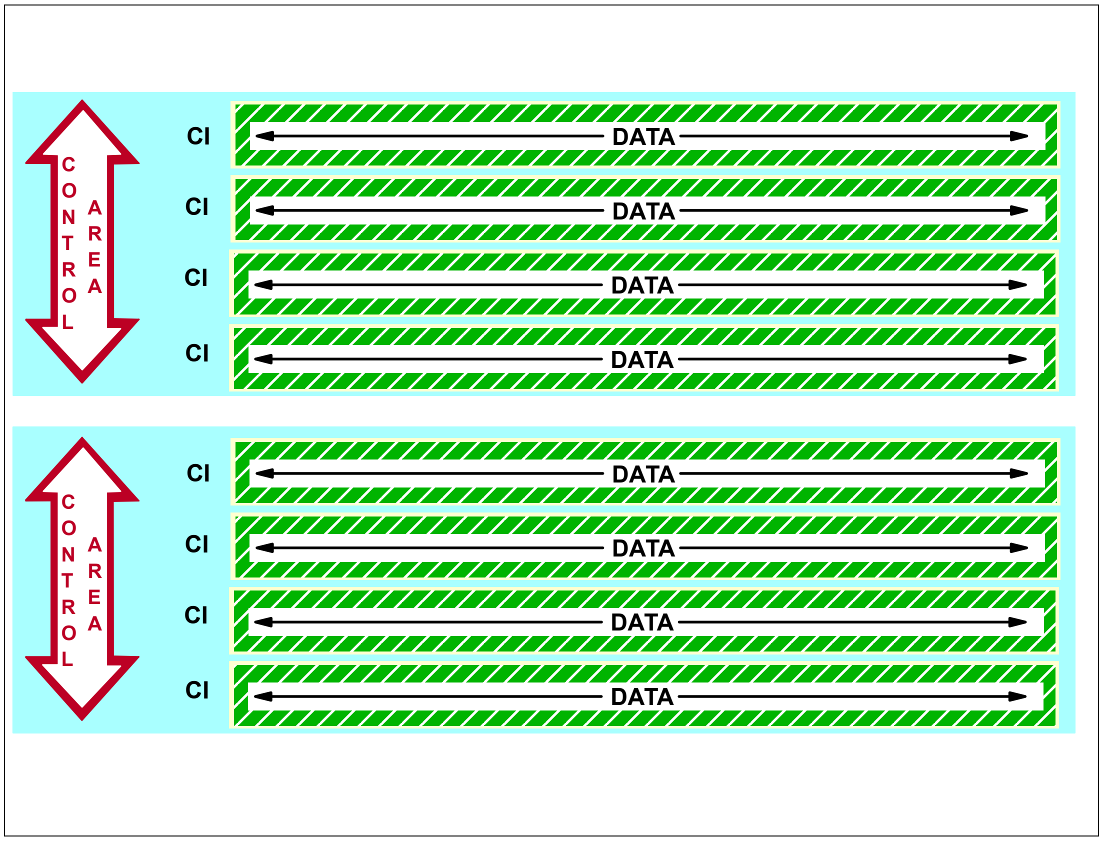


Figure 4-49 VSAM linear data set (LDS)

VSAM linear data set (LDS)

A *linear data set* is a VSAM data set with a control interval size from 4096 bytes to 32768 bytes, in increments of 4096 bytes. An LDS has no imbedded control information in its CI, that is, no record definition fields (RDFs) and no control interval definition fields (CIDFs). Therefore, all LDS bytes are *data* bytes. Logical records must be blocked and deblocked by the application program—but logical records do not exist from the point of view of VSAM.

IDCAMS is used to define a linear data set. An LDS has only a data component. An LDS data set is just a physical sequential VSAM data set comprised of 4 KB physical records, but with a revolutionary buffer technique called data-in-virtual (DIV).

A linear data set is processed as an entry-sequenced data set, with certain restrictions. Because a linear data set does not contain control information, it cannot be accessed as though it contained individual records. You can access a linear data set with the DIV macro. If using DIV to access the data set, the control interval size must be 4096; otherwise, the data set will not be processed.

When a linear data set is accessed with the DIV macro, it is referred to as the *data-in-virtual object* or the *data object*.

For information about how to use data-in-virtual, see *z/OS MVS Programming: Assembler Services Guide,* SA22-7605.

4.40 VSAM: Data-in-virtual (DIV)

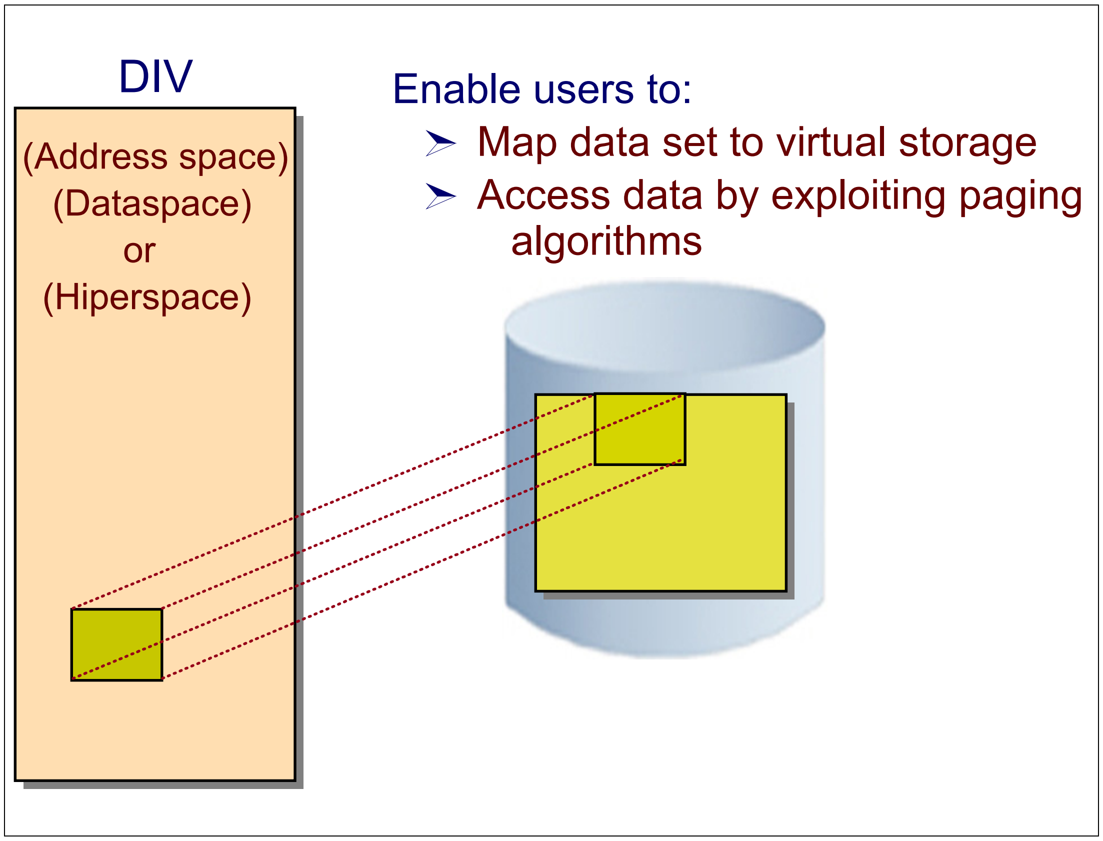

DIV
(Address space)
(Dataspace)
or
(Hiperspace)
Enable users to:
Map data set to virtual storage
Access data by exploiting paging algorithms

Figure 4-50 Data-in-virtual (DIV)

Data-in-virtual (DIV)

You can access a linear data set using these techniques:

- VSAM
- DIV, if the control interval size is 4096 bytes. The data-in-virtual (DIV) macro provides access to VSAM linear data sets.
- Window services, if the control interval size is 4096 bytes.

Data-in-virtual (DIV) is an optional and unique buffering technique used for LDS data sets. Application programs can use DIV to *map* a data set (or a portion of a data set) into an address space, a data space, or a hiperspace. An LDS cluster is sometimes referred to as a *DIV object*. After setting the environment, the LDS cluster looks to the application as a table in virtual storage with no need of issuing I/O requests.

Data is read into central storage via the paging algorithms only when that block is actually referenced. During RSM™ page-steal processing, only changed pages are written to the cluster in DASD. Unchanged pages are discarded since they can be retrieved again from the permanent data set.

DIV is designed to improve the performance of applications that process large files non-sequentially and process them with significant locality of reference. It reduces the number of I/O operations that are traditionally associated with data retrieval. Likely candidates are large arrays or table files.

4.41 VSAM: Mapping a linear data set

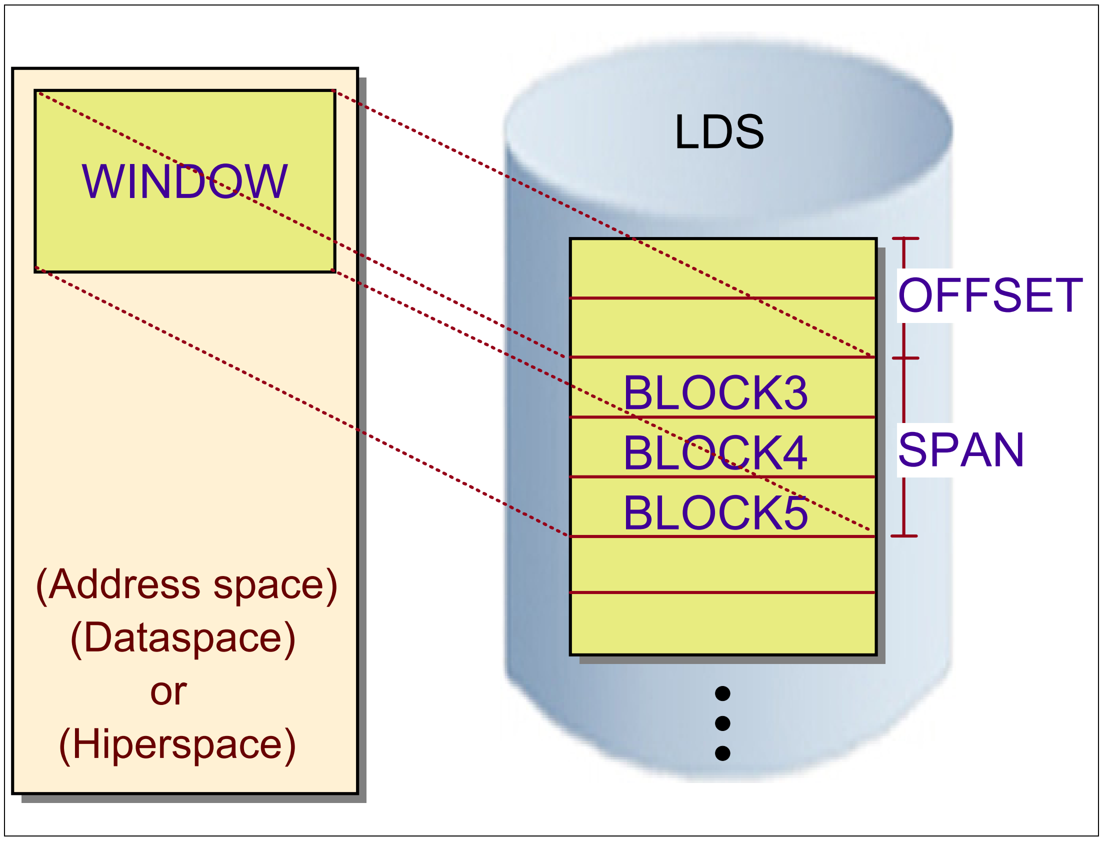


Figure 4-51 Mapping a linear data set

Mapping a linear data set

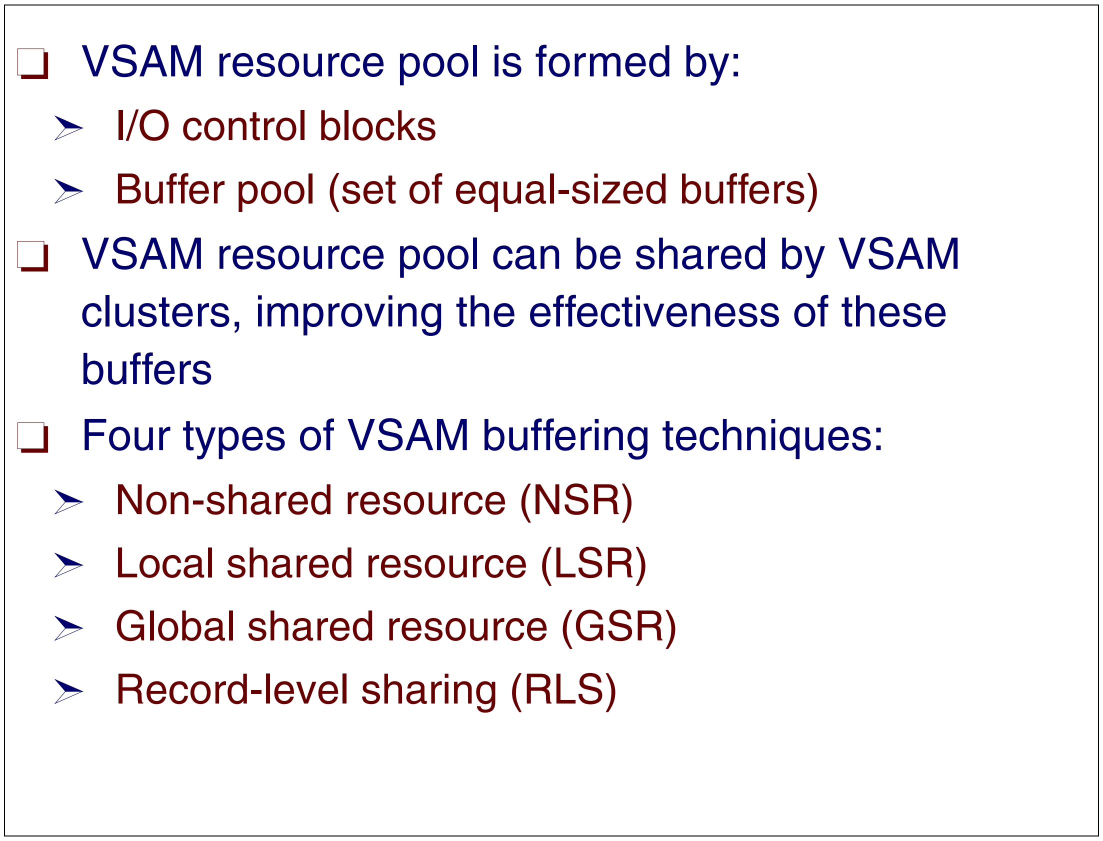
To establish a map from a linear data set to a window (a program-provided area in multiples of 4 KB on a 4 KB boundary), the program issues:

- DIV IDENTIFY to introduce (allocate) a linear data set to data-in-virtual services.
- DIV ACCESS to cause a VSAM open for the data set and indicate access mode (read/update).
- GETMAIN to allocate a window in virtual storage where the LDS will be mapped totally or in pieces.
- DIV MAP to enable the viewing of the data object by establishing an association between a program-provided area (window) and the data object. The area may be in an address space, data space, or hiperspace.

No actual I/O is done until the program references the data in the window. The reference will result in a page fault which causes data-in-virtual services to read the data from the linear data set into the window.

DIV SAVE can be used to write out changes to the data object. DIV RESET can be used to discard changes made in the window since the last SAVE operation.

4.42 VSAM resource pool



- ❑ VSAM resource pool is formed by:
  - ➢ I/O control blocks
  - ➢ Buffer pool (set of equal-sized buffers)
- ❑ VSAM resource pool can be shared by VSAM clusters, improving the effectiveness of these buffers
- ❑ Four types of VSAM buffering techniques:
  - ➢ Non-shared resource (NSR)
  - ➢ Local shared resource (LSR)
  - ➢ Global shared resource (GSR)
  - ➢ Record-level sharing (RLS)

Figure 4-52 VSAM resource pool

VSAM resource pool

Buffering is one of the key aspects as far as I/O performance is concerned. A VSAM *buffer* is a virtual storage area where the CI is transferred during an I/O operation. In VSAM KSDS there are two types of buffers: buffers for data CIs and buffers for index CIs. A *buffer pool* is a set of buffers with the same size. A *resource pool* is a buffer pool with some control blocks describing the pool and describing the clusters with CIs in the resource pool.

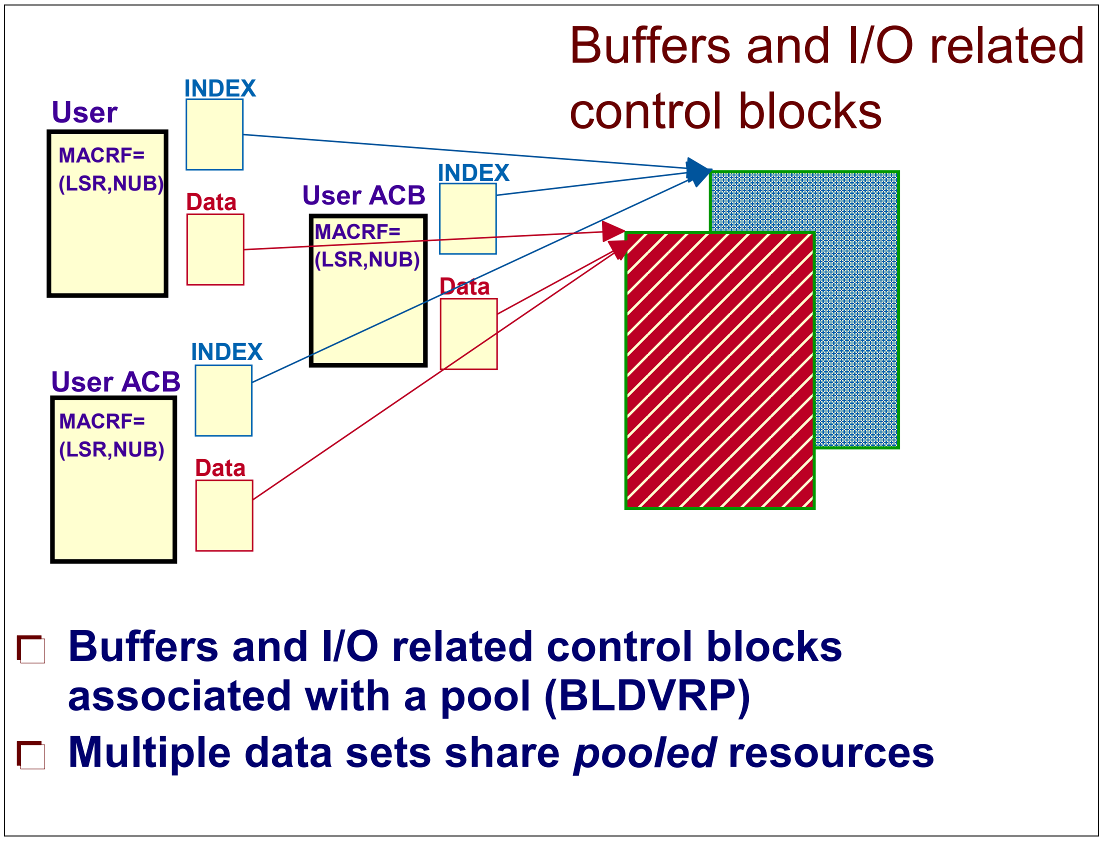
The objective of a buffer pool is to avoid I/O operations in random accesses (due to re-visiting data) and to make these I/O operations more efficient in sequential processing, thereby improving performance.

For more efficient use of virtual storage, buffer pools can be shared among clusters using locally or globally shared buffer pools. There are four types of resource pool management, called *modes*, defined according to the technique used to manage them:

- Not shared resources (NSR)
- Local shared resources (LSR)
- Global shared resources (GSR)
- Record-level shared resources (RLS)

These modes can be declared in the ACB macro of the VSAM data set (MACRF keyword) and are described in the following section.

4.43 VSAM: Buffering modes



Buffers and I/O related control blocks

- Buffers and I/O related control blocks associated with a pool (BLDVRP)
- Multiple data sets share *pooled* resources

Figure 4-53 VSAM LSR buffering mode

VSAM buffering modes

The VSAM buffering modes that you can use are:

- NSR, LSR, GSR, and RLS

Non-shared resource (NSR)

Non-shared resource (NSR) is the default VSAM buffering technique. It has the following characteristics:

- The resource pool is implicitly constructed at data set open time.
- The buffers are not shared among VSAM data sets; only one cluster has CIs in this resource pool.
- Buffers are located in the private area.
- For sequential reads, VSAM uses the read-ahead function: when the application finishes processing half the buffers, VSAM schedules an I/O operation for that half of the buffers.

  This continues until a CA boundary is encountered; the application must wait until the last I/O to the CA is done before proceeding to the next CA. The I/O operations are always scheduled within CA boundaries.
- For sequential writes, VSAM postpones the writes to DASD until half the buffers are filled by the application. Then VSAM schedules an I/O operation to write that half of the buffers to DASD. The I/O operations are always scheduled within CA boundaries.

- CIs are discarded as soon as they are used, using a sequential algorithm to keep CIs in the resource pool.
- There is dynamic addition of strings. Strings are like cursors; each string represents a position in the data set for the requested record.
- For random access there is not look ahead, but the algorithm is still the sequential one.

NSR is used by high-level languages. Since buffers are managed via a sequential algorithm, NSR is not the best choice for random processing. For applications using NSR, consider using system-managed buffering, discussed in “VSAM: System-managed buffering (SMB)” on page 182.

Local shared resource (LSR)

An LSR resource pool is suitable for random processing and not for sequential processing. The characteristics of the LSR include that it is:

- Shared among VSAM clusters accessed by tasks in the same address space.
- Located in the private area and ESO hiperspace. With hiperspace, VSAM buffers are located in expanded storage to improve the processing of VSAM clusters. With z/Architecture ESO hiperspaces are mapped in central storage.
- A resource pool is explicitly constructed via macro BLDVRP, before the OPEN.
- Buffers are managed via the last recently used (LRU) algorithm, that is, the most referenced CIs are kept in the resource pool. It is very adequate for random processing.
- LSR expects that CIs in buffers *are* re-visited.

Global shared resource (GSR)

GSR is similar to the LSR buffering technique. GSR differs from LSR in the following ways:

- The buffer pool is shared among VSAM data sets accessed by tasks in *multiple* address spaces in the same z/OS image.
- Buffers are located in CSA.
- The code using GSR must be in the supervisor state.
- Buffers cannot use hiperspace.
- The separate index resource pools are not supported for GSR.

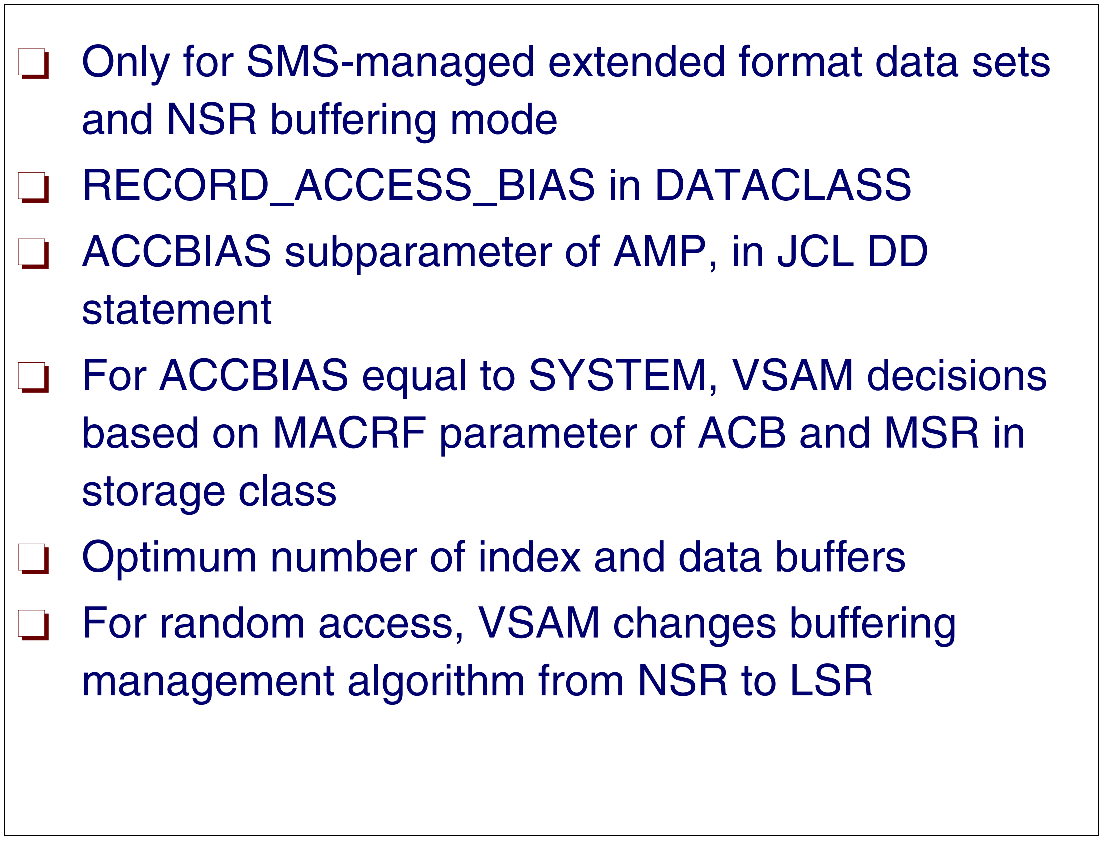
GSR is not commonly used by applications, so you should consider the use of VSAM RLS instead.

Record-level sharing (RLS)

This is the implementation of VSAM data sharing. Record-level sharing is discussed in detail in “DFSMS Transactional VSAM Services” on page 353.

For more information about NSR, LSR, and GSR, refer to “Base VSAM buffering” on page 356 and also to the IBM Redbooks document *VSAM Demystified,* SG24-6105.

4.44 VSAM: System-managed buffering (SMB)



- Only for SMS-managed extended format data sets and NSR buffering mode
- RECORD_ACCESS_BIAS in DATACLASS
- ACCBIAS subparameter of AMP, in JCL DD statement
- For ACCBIAS equal to SYSTEM, VSAM decisions based on MACRF parameter of ACB and MSR in storage class
- Optimum number of index and data buffers
- For random access, VSAM changes buffering management algorithm from NSR to LSR

Figure 4-54 System-managed buffering (SMB)

System-managed buffering (SMB)

SMB is a feature of VSAM that was introduced in DFSMS V1R4. SMB enables VSAM to:

- Determine the optimum number of index and data buffers
- Change the buffer algorithm as declared in the application program in the ACB MACRF parameter, from NSR (sequential) to LSR (least recently used - LRU)

Usually, SMB allocates many more buffers than would be allocated without SMB. Performance improvements can be dramatic with random access (particularly when few buffers were available). The use of SMB is transparent from the point of view of the application; no application changes are needed.

SMB is available to a data set when *all* the following conditions are met:

- It is an SMS-managed data set.
- It is in extended format (DSNTYPE = EXT in the data class).
- The application opens the data set for NSR processing.

SMB is invoked or disabled through one of the following methods:

1. The Record Access Bias data class field.
2. The ACCBIAS subparameter of AMP in the JCL DD statement. JCL information takes precedence over data class information.

If all of the required conditions are met, SMB is invoked when option SYSTEM or an SMB processing technique is used in the fields described previously. SMB is disabled when USER is entered instead (USER is the default). Since JCL information takes precedence over data class information, installations can enable or disable SMB for some executions.

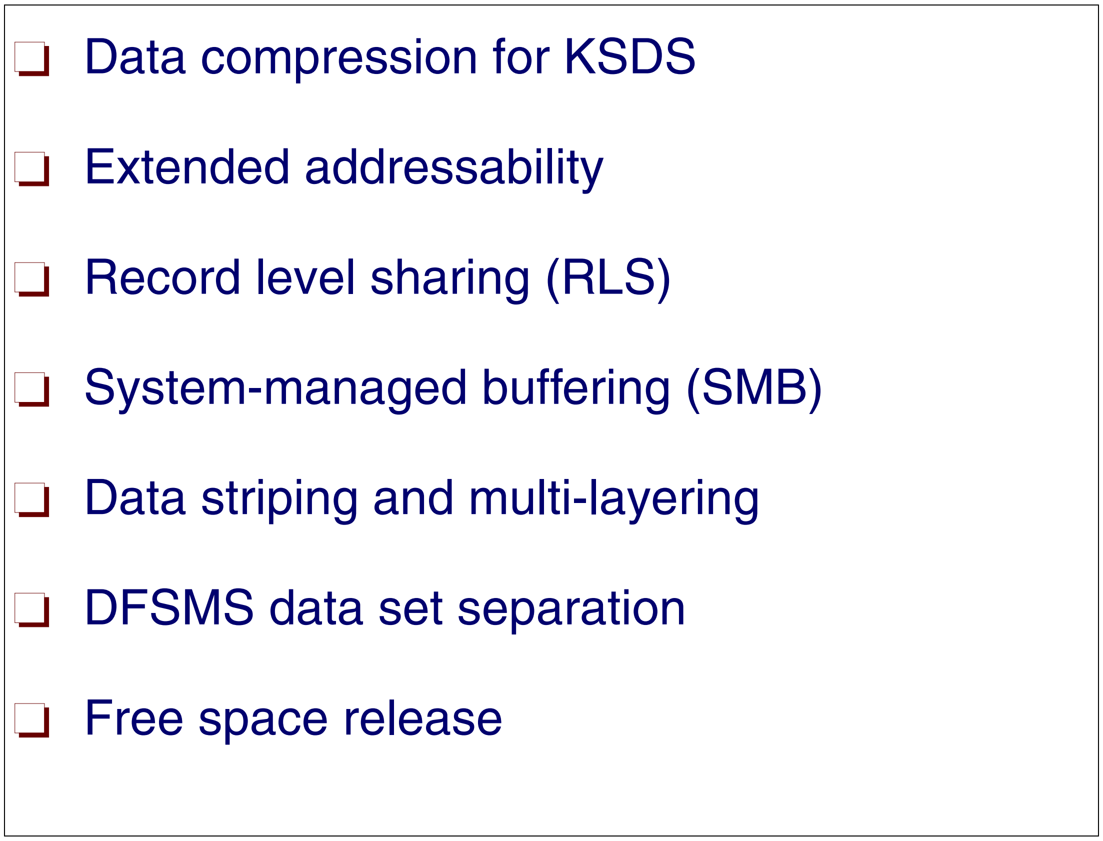
The information contained in SMB processing techniques is needed because SMB must maintain an adequate algorithm for managing the CIs in the resource pool. SMB accepts the ACB MACRF options when the I/O operation is requested. For this reason, the installation must accurately specify the processing type, through ACCBIAS options:

- *Direct Optimized (DO)*: SMB optimizes for totally random record access. When this technique is used, VSAM changes the buffering management from NSR to LSR.
- *Direct Weighted (DW)*: The majority is direct access to records, with some sequential.
- *Sequential Optimized (SO)*: Totally sequential access.
- *Sequential Weighted (SW)*: The majority is sequential access, with some direct access to records.

When SYSTEM is used in JCL or in the data class, SMB chooses the processing technique based on the MACRF parameter of the ACB.

For more information about the use of SMB, refer to *VSAM Demystified,* SG24-6105.

4.45 VSAM enhancements



- Data compression for KSDS
- Extended addressability
- Record level sharing (RLS)
- System-managed buffering (SMB)
- Data striping and multi-layering
- DFSMS data set separation
- Free space release

Figure 4-55 VSAM enhancements

VSAM enhancements

Following is a list of the major VSAM enhancements since DFSMS V1R2. For the majority of these functions extended format is a prerequisite. The enhancements are:

- Data compression for KSDS: Good for improving I/O mainly for write-once, read-many clusters.
- Extended addressability: Allows data components larger than 4 GB. The limitation was caused by an RBA field of 4 byte, now RBA has an 8 byte length.
- Record-level sharing (RLS): Allows VSAM data sharing across z/OS systems in a Parallel Sysplex.
- System-managed buffering (SMB): Improves the performance of random NSR processing.
- Data stripping and multi-layering: Improves sequential access performance due to parallel I/Os in several volumes (stripes).
- DFSMS data set separation: Allows the allocation of cluster is distinct physical control units.
- Free space release: As for non-VSAM data sets, the free space not used at the end of the data component can be released at de-allocation.

4.46 Data facility sort (DFSORT)

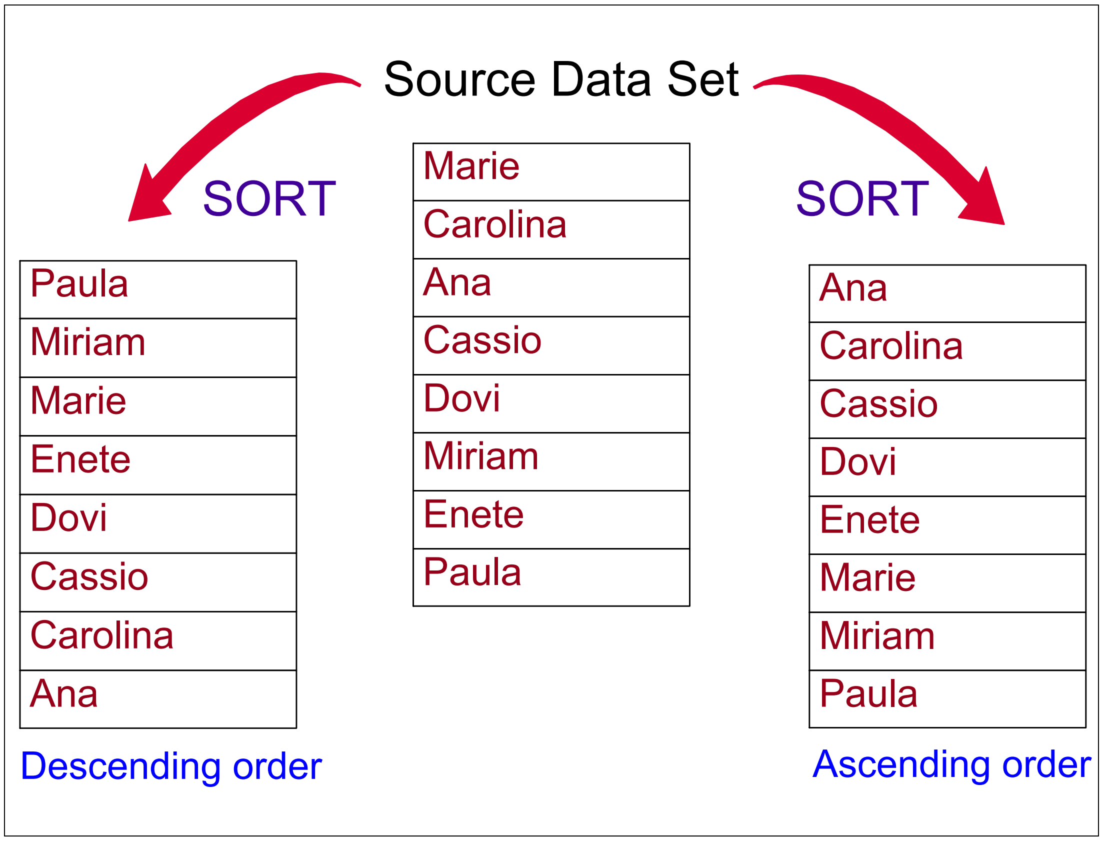


Figure 4-56 DFSORT example

Data facility sort (DFSORT)

The DFSORT licensed program is a high performance data arranger for z/OS users. With DFSORT, you can sort, merge, and copy data sets using EBCDIC, z/architecture decimal or binary keys. DFSORT is an optional feature of z/OS.

DFSORT, together with DFSMS and RACF, form the strategic product base for the evolving system-managed storage environment. DFSORT is designed to optimize the efficiency and speed with which operations are completed through synergy with processor, device, and system features (for example, memory objects, Hiperspace™, data space, striping, compression, extended addressing, DASD and tape device architecture, processor memory, and processor cache.

DFSORT example

The simple example in Figure 4-56 illustrates how DFSORT merges data sets by combining two or more files of sorted records to form a single data set of sorted records.

You can use DFSORT to do simple application tasks such as alphabetizing a list of names, or you can use it to aid complex tasks such as taking inventory or running a billing system. You can also use DFSORT's record-level editing capability to perform data management tasks.

For most of the processing done by DFSORT, the whole data set is affected. However, some forms of DFSORT processing involve only certain individual records in that data set.

DFSORT functions

DFSORT adds the ability to do faster and easier sorting, merging, copying, reporting and analysis of your business information, as well as versatile data handling at the record, fixed position/length or variable position/length field, and bit level. While sorting, merging, or copying data sets, you can also:

- Select a subset of records from an input data set. You can include or omit records that meet specified criteria. For example, when sorting an input data set containing records of course books from many different school departments, you can sort the books for only one department.
- Reformat records, add or delete fields, and insert blanks, constants, or binary zeros. For example, you can create an output data set that contains only certain fields from the input data set, arranged differently.
- Sum the values in selected records while sorting or merging (but not while copying). In the example of a data set containing records of course books, you can use DFSORT to add up the dollar amounts of books for one school department.
- Create multiple output data sets and reports from a single pass over an input data set. For example, you can create a different output data set for the records of each department.
- Sort, merge, include, or omit records according to the collating rules defined in a selected local.
- Alter the collating sequence when sorting or merging records (but not while copying). For example, you can have the lowercase letters collate after the uppercase letters.
- Sort, merge, or copy Japanese data if the IBM Double Byte Character Set Ordering Support (DBCS Ordering, the 5665-360 Licensed Program, Release 2.0 or an equivalent product) is used with DFSORT to process the records.

DFSORT has utilities such as ICETOOL, which is a multipurpose DFSORT utility that uses the capabilities of DFSORT to perform multiple operations on one or more data sets in a single step.

Specifying the DFSORT customization parameters is a very important task for the z/OS system programmers. Depending on such parameters, DFSORT may use lots of system resources such as CPU, I/O, and especially virtual storage. The uncontrolled use of virtual storage may cause IPLs due to the lack of available slots in page data sets. Plan to use the IEFUSI z/OS exit to control products such as DFSORT.

For articles, online books, news, tips, techniques, examples, and more, visit the z/OS DFSORT home page:

```
http://www-1.ibm.com/servers/storage/support/software/sort/mvs
```

4.47 z/OS Network File System (z/OS NFS)

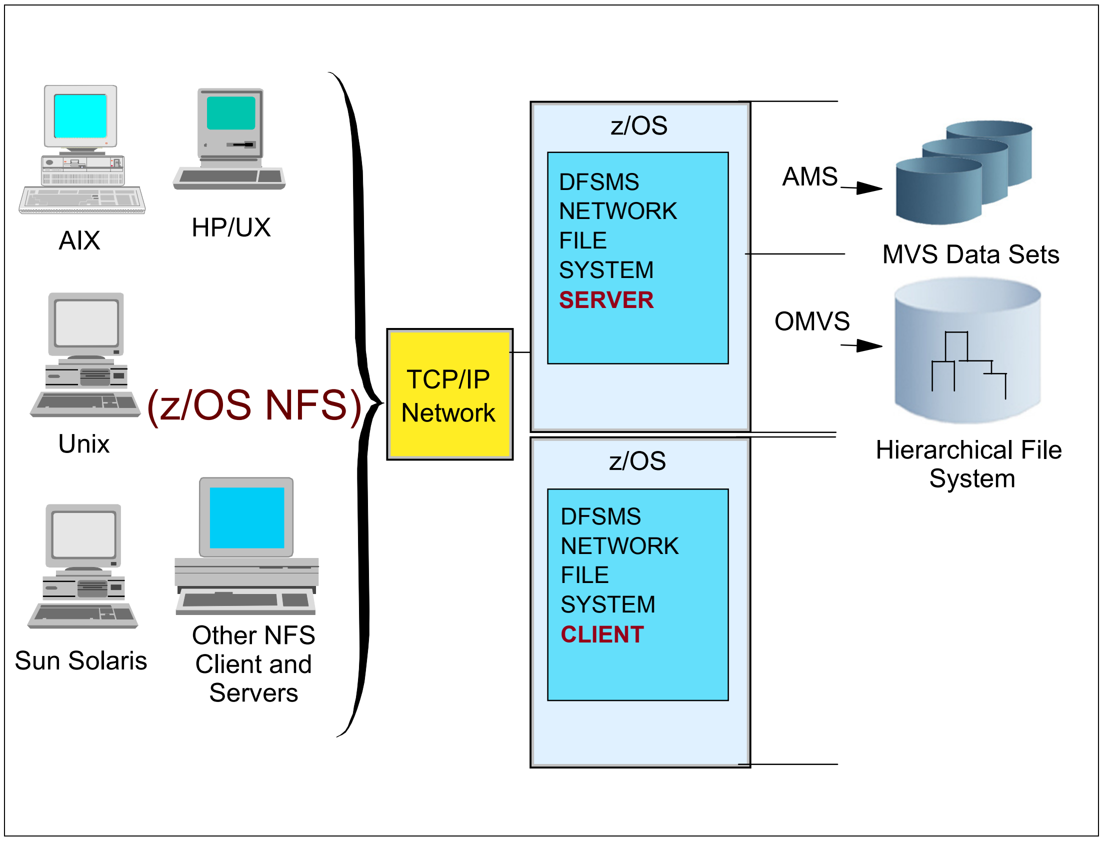


Figure 4-57 DFSMS Network File System

z/OS Network File System (z/OS NFS)

The z/OS Network File System is a distributed file system that enables users to access UNIX files and directories that are located on remote computers as though they were local. NFS is independent of machine types, operating systems, and network architectures. Use the NFS for file serving (as a data repository) and file sharing between platforms supported by z/OS.

Clients and servers

A *client* is a computer or process that requests services on the network. A *server* is a computer or process that responds to a request for service from a client. A user accesses a *service*, which allows the use of data or other resources.

Figure 4-57 illustrates the client-server relationship:

- The upper center portion shows the DFSMS NFS address space server; the lower portion shows the DFSMS NFS address space client.
- The left side of the figure shows various NFS clients and servers that can interact with the DFSMS NFS server and client.
- In the center of the figure is the Transmission Control Protocol/Internet Protocol (TCP/IP) network used to communicate between clients and servers.

With the NFS server, you can remotely access z/OS conventional data sets or z/OS UNIX files from workstations, personal computers, and other systems that run client software for the

Sun NFS version 2 protocols, the Sun NFS version 3 protocols, and the WebNFS protocols over TCP/IP network.

The z/OS NFS server acts as an intermediary to read, write, create, or delete z/OS UNIX files and MVS data sets that are maintained on an MVS host system. The remote MVS data sets or z/OS UNIX files are mounted from the host processor to appear as local directories and files on the client system.

This server makes the strengths of a z/OS host processor—storage management, high-performance disk storage, security, and centralized data—available to the client platforms.

With the NFS client you can allow basic sequential access method (BSAM), queued sequential access method (QSAM), virtual storage access method (VSAM), and z/OS UNIX users and applications transparent access to data on systems that support the Sun NFS version 2 protocols and the Sun NFS version 3 protocols.

The Network File System can be used for:

- File sharing between platforms
- File serving (as a data repository)

Supported clients for the NFS server

The z/OS NFS client supports all servers that implement the server portion of the Sun NFS Version 2 and Version 3 protocols. The z/OS NFS client does not support NFS version 4. Tested clients for the z/OS NFS server, using the NFS version 4 protocol, are:

- IBM RS/6000 AIX version 5.3
- Sun Solaris version 10
- Enterprise Linux 4
- Windows 2000/XP with Hummingbird Maestro™ 9 and Maestro 10

Other client platforms should work as well since NFS version 4 is an industry standard protocol, but they have not been tested by IBM.

NFS client software for other IBM platforms is available from other vendors. You can also access the NFS server from non-IBM clients that use the NFS version 2 or version 3 protocol, including:

- DEC stations running DEC ULTRIX version 4.4
- HP 9000 workstations running HP/UX version 10.20
- Sun PC-NFS version 5
- Sun workstations running SunOS™ or Sun Solaris versions 2.5.3

For further information about NFS, refer to *z/OS Network File System Guide and Reference*, SC26-7417, and visit:

```
http://www-1.ibm.com/servers/eserver/zseries/zos/nfs/
```

4.48 DFSMS optimizer (DFSMSopt)

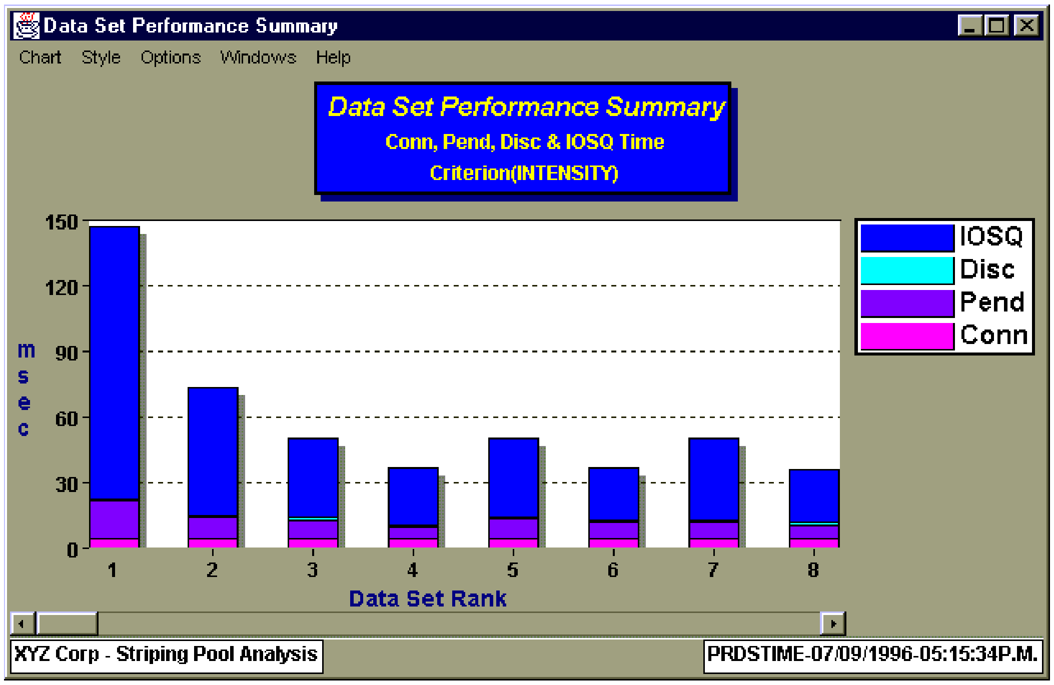

Data Set Performance Summary
Chart
Style
Options
Windows
Help
Data Set Performance Summary
Conn, Pend, Disc & IOSQ Time
Criterion(INTENSITY)
150
120
90
60
30
0
m s e c
1
2
3
4
5
6
7
8
Data Set Rank
IOSQ
Disc
Pend
Conn
XYZ Corp - Striping Pool Analysis
PRDSTIME-07/09/1996-05:15:34P.M.

Figure 4-58 DFSMS Optimizer: Data set performance summary

DFSMS Optimizer (DFSMSopt)

The DFSMS Optimizer gives you the ability to understand how you manage storage today. With that information, you can make informed decisions about how you should manage storage in the future.

These are the analyzers you can use:

- Performance analyzer
- Management class analyzer
- I/O trace analyzer

DFSMS Optimizer uses input data from several sources in the system and processes it using an extract program that merges the data and builds the Optimizer database.

By specifying different filters you can produce reports that help you build a detailed storage management picture of your enterprise. With the report data, you can use the charting facility to produce color charts and graphs.

The DFSMS Optimizer provides analysis and simulation information for both SMS and non-SMS users. The DFSMS Optimizer can help you maximize storage use and minimize storage costs. It provides methods and facilities for you to:

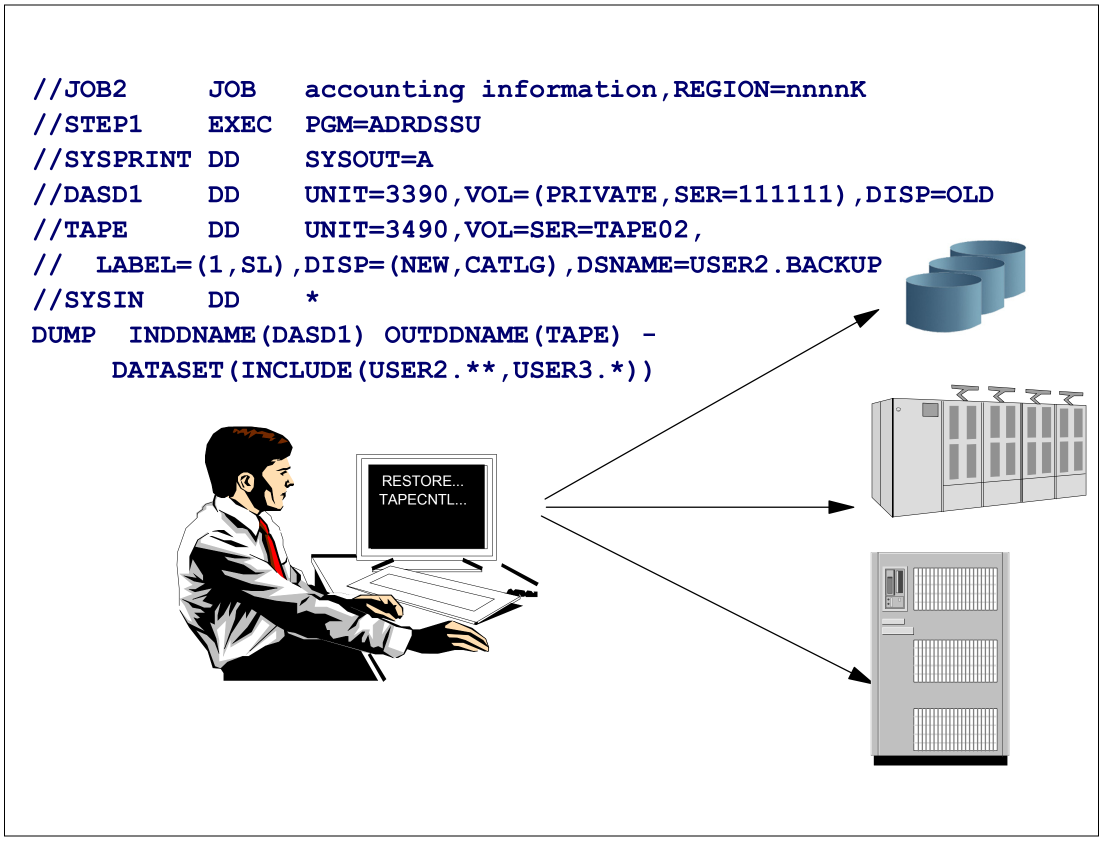
- Monitor and tune DFSMShsm functions such as migration and backup
- Create and maintain a historical database of system and data activity

- Fine tune an SMS configuration by performing in-depth analysis of:
  - Management class policies, including simulations and cost/benefit analysis using your storage component costs
  - Storage class policies for SMS data, with recommendations for both SMS and non-SMS data
  - High I/O activity data sets, including recommendations for placement and simulation for cache and expanded storage
  - Storage hardware performance of subsystems and volumes including I/O rate, response time, and caching statistics
- Simulate potential policy changes and understand the costs of those changes
- Produce presentation-quality charts

For more information about the DFSMS Optimizer, refer to *DFSMS Optimizer User's Guide and Reference,* SC26-7047 or visit:

```
http://www-1.ibm.com/servers/storage/software/opt/
```

4.49 Data Set Services (DFSMSdss)



```
//JOB2      JOB   accounting information,REGION=nnnnK
//STEP1     EXEC  PGM=ADRDSSU
//SYSPRINT DD     SYSOUT=A
//DASD1     DD    UNIT=3390,VOL=(PRIVATE,SER=111111),DISP=OLD
//TAPE      DD    UNIT=3490,VOL=SER=TAPE02,
//   LABEL=(1,SL),DISP=(NEW,CATLG),DSNAME=USER2.BACKUP
//SYSIN     DD    *
DUMP   INDDNAME(DASD1) OUTDDNAME(TAPE) -
     DATASET(INCLUDE(USER2.**,USER3.*))
```

Figure 4-59 DFSMSdss backing up and restoring volumes and data sets

Data Set Services (DFSMSdss)

DFSMSdss is a direct access storage device (DASD) data and space management tool. DFSMSdss works on DASD volumes only in a z/OS environment. You can use DFSMSdss to do the following:

- ► Copy and move data sets between volumes of like and unlike device types.

Note: Like devices have the same track capacity and number of tracks per cylinder (for example, 3380 Model D, Model E, and Model K). Unlike DASD devices have different track capacities (for example, 3380 and 3390), a different number of tracks per cylinder, or both.

- ► Dump and restore data sets, entire volumes, or specific tracks.
- ► Convert data sets and volumes to and from SMS management.
- ► Compress partitioned data sets.
- ► Release unused space in data sets.
- ► Reduce or eliminate DASD free-space fragmentation by consolidating free space on a volume.
- ► Implement concurrent copy or a flashcopy in ESS or DS8000 DASD controllers.

Backing up and restoring volumes and data sets with DFSMSdss

You can use the DFSMSdss DUMP command to back up volumes and data sets, and you can use the DFSMSdss RESTORE command to recover them. You can make incremental backups of your data sets by specifying a data set DUMP command with RESET and filtering on the data-set-changed indicator.

The DFSMShsm component of DFSMS provides automated incremental backup, interactive recovery, and inventory of what it backs up. If DFSMShsm is used, you should use DFSMSdss for volume backup of data sets not supported by DFSMShsm and for dumping SYSRES and special volumes such as the one containing the master catalog, as shown in Figure 4-59 on page 191. If DFSMShsm is not installed, you can use DFSMSdss for all volume and data set backups.

4.50 DFSMSdss: Physical and logical processing

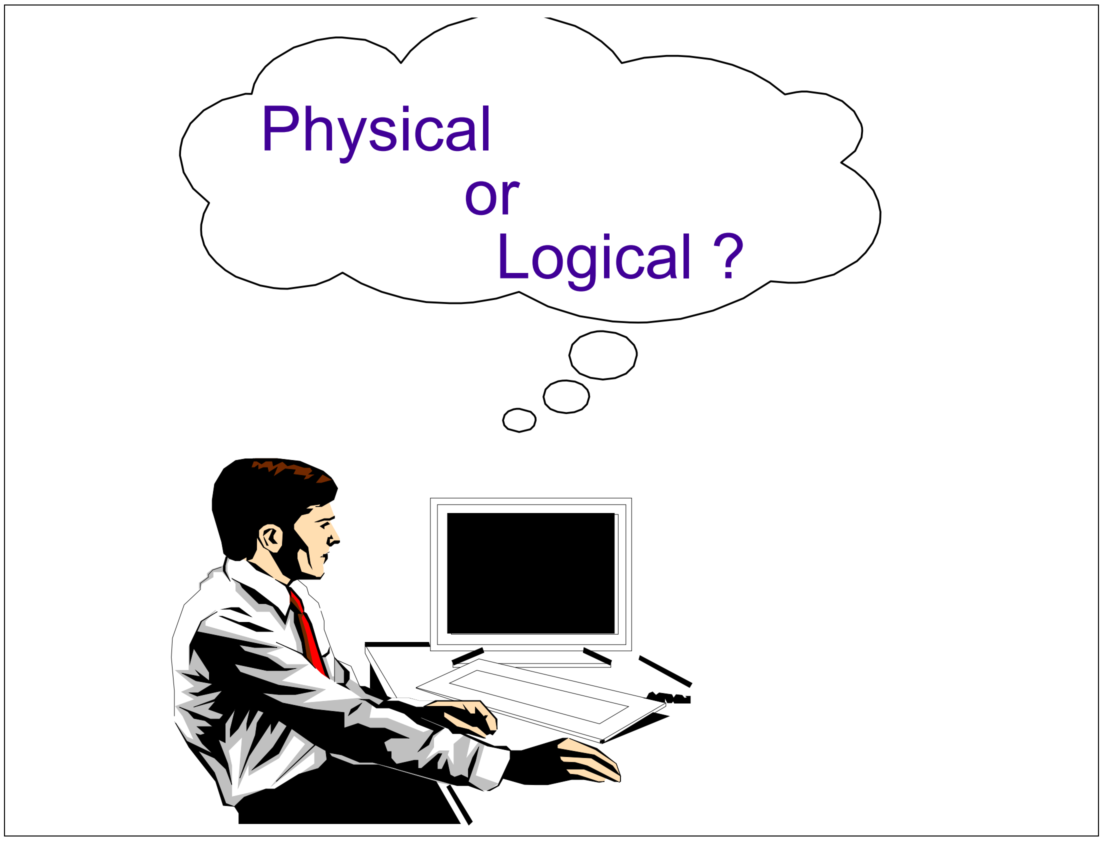


Figure 4-60 DFSMSdss physical and logical processing

DFSMSdss: physical and logical processing

Before you begin using DFSMSdss, you should understand the difference between logical processing and physical processing and how to use data set filtering to select data sets for processing. DFSMSdss can perform two kinds of processing when executing **COPY**, **DUMP**, and **RESTORE** commands:

- *Logical processing* operates against data sets independently of physical device format.
- *Physical processing* moves data at the track-image level and operates against volumes, tracks, and data sets.

Each type of processing offers different capabilities and advantages.

During a restore operation, the data is processed the same way it is dumped because physical and logical dump tapes have different formats. If a data set is dumped logically, it is restored logically; if it is dumped physically, it is restored physically. A data set restore operation from a full volume dump is a physical data set restore operation.

4.51 DFSMSdss: Logical processing

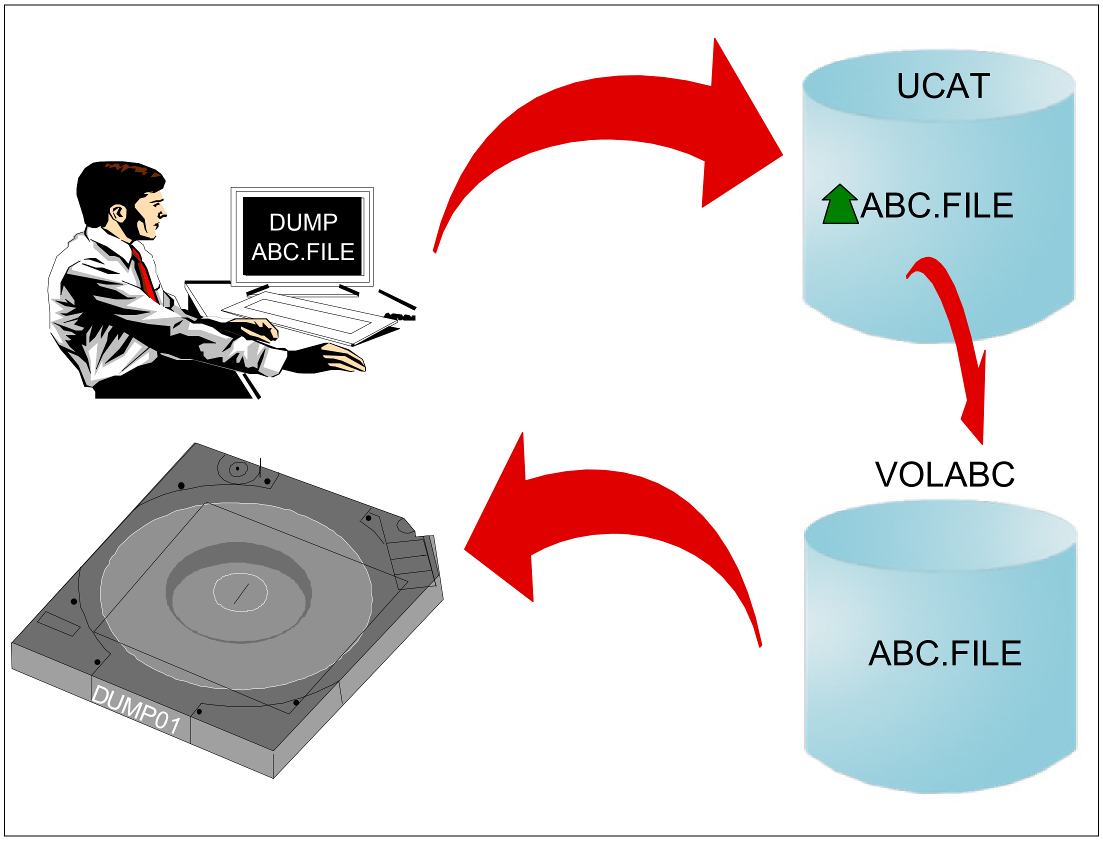


Figure 4-61 DFSMSdss logical processing

Logical processing

A logical copy, dump, or restore operation treats each data set and its associated information as a logical entity, and processes an entire data set before beginning the next one.

Each data set is moved by tracks from the source device and is potentially written to the target device as a set of data records, allowing data movement between devices with different track and cylinder configurations. Checking of data record consistency is not performed during dump operation.

DFSMSdss performs logical processing if:

- You specify the DATASET keyword with the **COPY** command. A data set copy is always a logical operation, regardless of how or whether you specify input volumes.
- You specify the DATASET keyword with the **DUMP** command, and either no input volume is specified, or LOGINDDNAME, LOGINDYNAM, or STORGRP is used to specify input volumes.
- The **RESTORE** command is performed, and the input volume was created by a logical dump.

Catalogs and VTOCs are used to select data sets for logical processing. If you do not specify input volumes, the catalogs are used to select data sets for copy and dump operations. If you specify input volumes using the LOGINDDNAME, LOGINDYNAM, or STORGRP keywords on the COPY or DUMP command, DFSMSdss uses VTOCs to select data sets for processing.

When to use logical processing

Use logical processing for the following situations:

- Data is copied to an unlike device type.

  Logical processing is the only way to move data between unlike device types.
- Data that may need to be restored to an unlike device is dumped.
- Data must be restored the same way it is dumped.

  This is particularly important to bear in mind when making backups that you plan to retain for a long period of time (such as vital records backups). If a backup is retained for a long period of time, it is possible that the device type it originally resided on will no longer be in use at your site when you want to restore it. This means you will have to restore it to an unlike device, which can be done only if the backup has been made logically.
- Aliases of VSAM user catalogs are to be preserved during copy and restore functions.

  Aliases are not preserved for physical processing.
- Unmovable data sets or data sets with absolute track allocation are moved to different locations.
- Multivolume data sets are processed.
- VSAM and multivolume data sets are to be cataloged as part of DFSMSdss processing.
- Data sets are to be deleted from the source volume after a successful dump or copy operation.
- Both non-VSAM and VSAM data sets are to be renamed after a successful copy or restore operation.
- You want to control the percentage of space allocated on each of the output volumes for copy and restore operations.
- You want to copy and convert a PDS to a PDSE or vice versa.
- You want to copy or restore a data set with an undefined DSORG to an unlike device.
- You want to keep together all parts of a VSAM sphere.

4.52 DFSMSdss: Physical processing

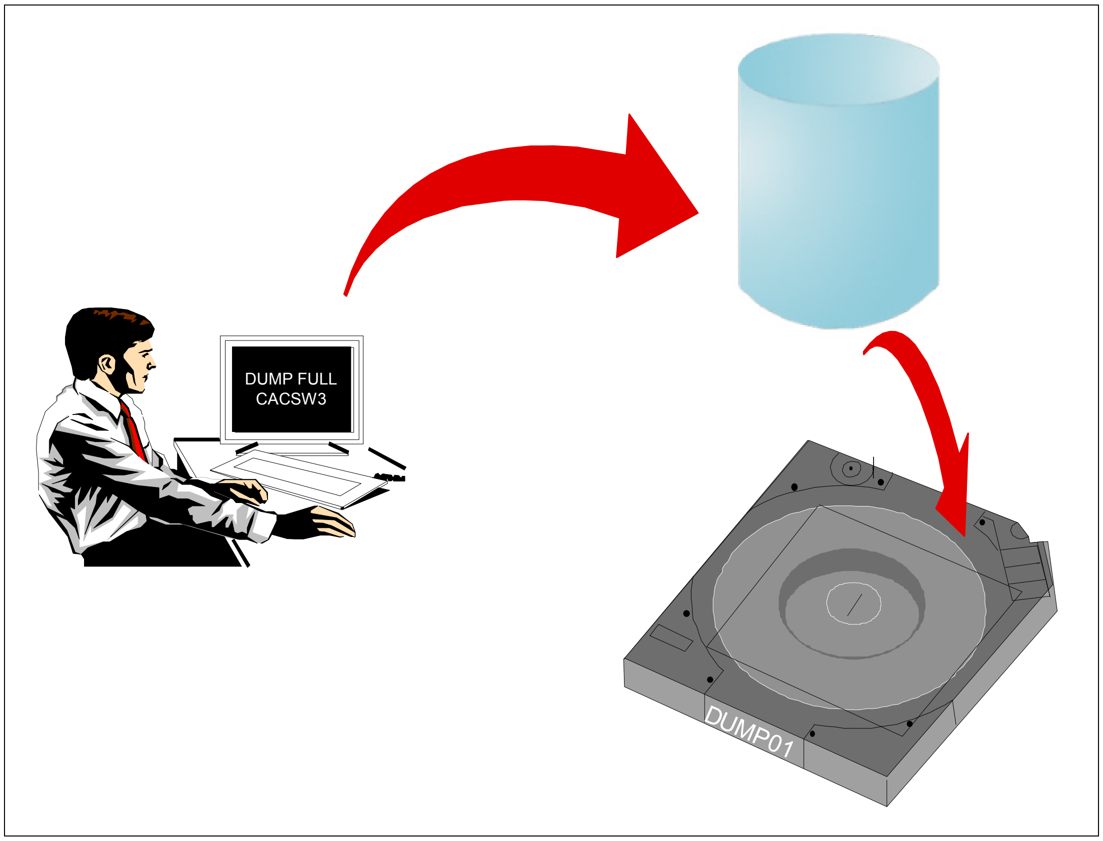


Figure 4-62 DFSMSdss physical processing

Physical processing

Physical processing moves data based on physical track images. Because data movement is carried out at the track level, only target devices with track sizes equal to those of the source device are supported. Physical processing operates on volumes, ranges of tracks, or data sets. For data sets, it relies only on volume information (in the VTOC and VVDS) for data set selection, and processes only that part of a data set residing on the specified input volumes.

DFSMSdss performs physical processing if:

- You specify the FULL or TRACKS keyword with the `COPY` or `DUMP` command. This results in a physical volume or physical tracks operation.

  Attention: Take care when invoking the TRACKS keyword with the `COPY` and `RESTORE` commands. The TRACKS keyword should be used only for a data recovery operation. For example, you can use it to “repair” a bad track in the VTOC or a data set, or to retrieve data from a damaged data set. You cannot use it in place of a full-volume or a logical data set operation. Doing so could destroy a volume or impair data integrity.

- You specify the data set keyword on the `DUMP` command and input volumes with the INDDNAME or INDYNAM parameter. This produces a physical data set dump.
- The `RESTORE` command is executed and the input volume is created by a physical dump operation.

When to use physical processing

Use physical processing when:

- Backing up system volumes that you might want to restore with a stand-alone DFSMSdss restore operation.

  Stand-alone DFSMSdss restore supports only physical dump tapes.
- Performance is an issue.

  Generally, the fastest way—measured by elapsed time—to copy or to dump an entire volume is with a physical full-volume command. This is primarily because minimal catalog searching is necessary for physical processing.
- Substituting one physical volume for another or recovering an entire volume.

  With a **COPY** or **RESTORE** (full volume or track) command, the volume serial number of the input DASD volume can be copied to the output DASD volume.
- Dealing with I/O errors.

  Physical processing provides the capability to copy, dump, and restore a specific track or range of tracks.
- Dumping or copying between volumes of the same device type but different capacity.

4.53 DFSMSdss stand-alone services

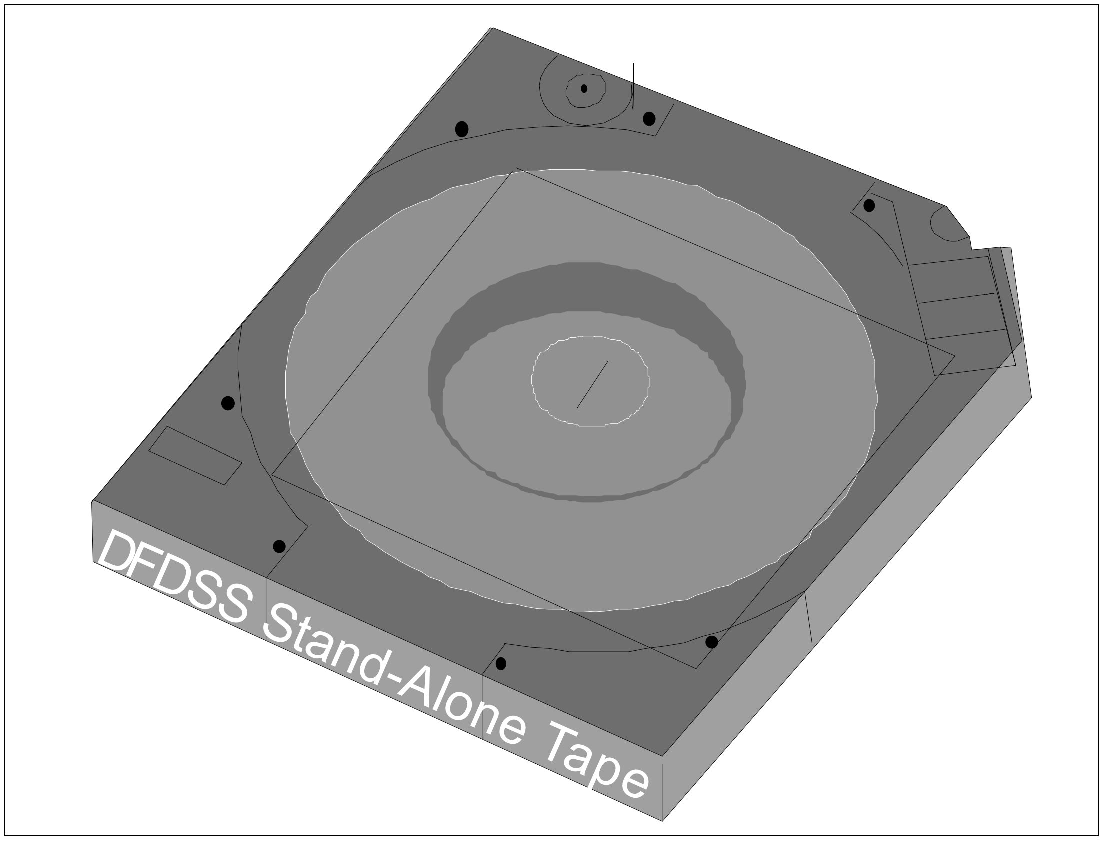


Figure 4-63 DFSMSdss stand-alone services

DFSMSdss stand-alone services

The DFSMSdss stand-alone restore function is a single-purpose program. It is designed to allow the system programmer to restore vital system packs during disaster recovery without relying on an MVS environment.

Stand-alone services can perform either a full-volume restore or a tracks restore from dump tapes produced by DFSMSdss or DFDSS and offers the following benefits:

- Provides user-friendly commands to replace the previous control statements
- Supports IBM 3494 and 3495 Tape Libraries, and 3590 Tape Subsystems
- Supports IPLing from a DASD volume, in addition to tape and card readers
- Allows you to predefine the operator console to be used during stand-alone services processing

For detailed information about the stand-alone service, and other DFSMSdss information, refer to *z/OS DFSMSdss Storage Administration Reference,* SC35-0424, and *z/OS DFSMSdss Storage Administration Guide*, SC35-0423 and visit:

```
http://www-1.ibm.com/servers/storage/software/sms/dss/
```

4.54 Hierarchical Storage Manager (DFSMShsm)

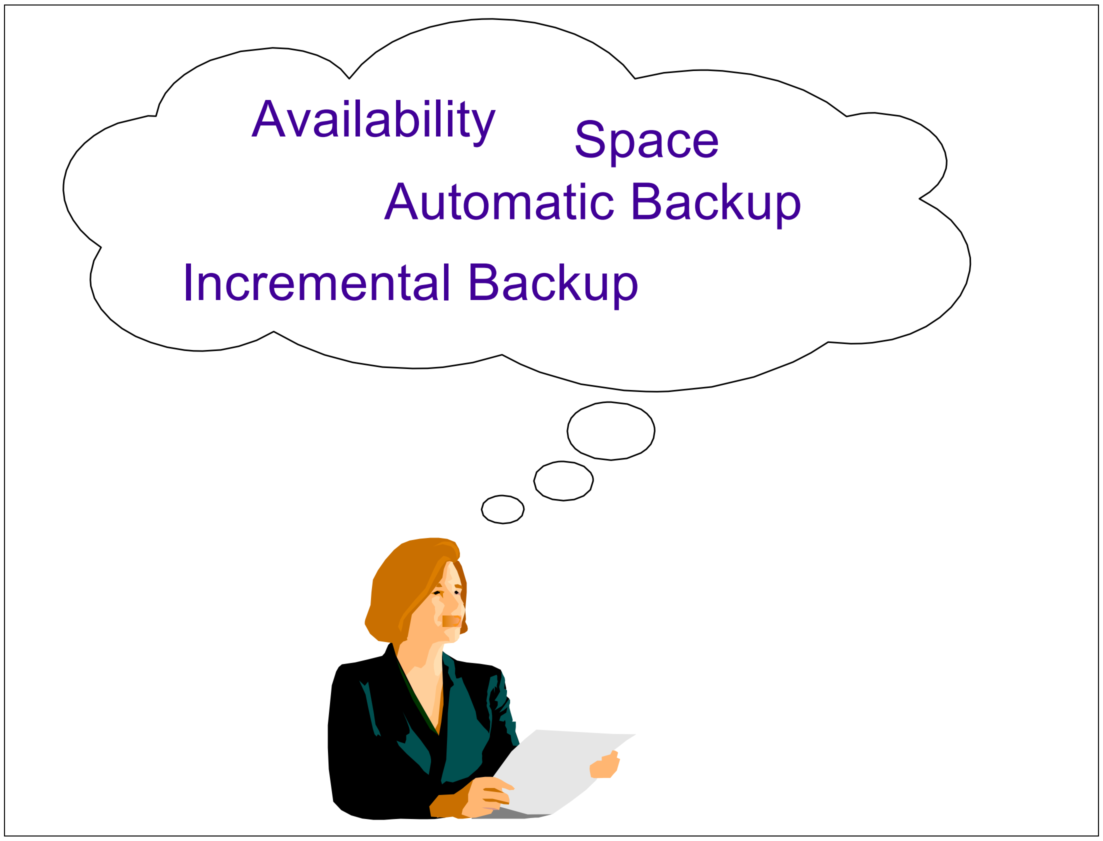


Figure 4-64 DFSMShsm

Hierarchical Storage Manager (DFSMShsm)

Hierarchical Storage Manager (DFSMShsm) is a disk storage management and productivity product for managing low activity and inactive data. It provides backup, recovery, migration, and space management functions as well as full-function disaster recovery support. DFSMShsm improves disk use by automatically managing both space and data availability in a storage hierarchy.

Availability management is used to make data available by automatically copying new and changed data set to backup volumes.

Space management is used to manage DASD space by enabling inactive data sets to be moved off fast-access storage devices, thus creating free space or new allocations.

DFSMShsm also provides for other supporting functions that are essential to your installation's environment.

For further information about DFSMShsm, refer to *z/OS DFSMShsm Storage Administration Guide,* SC35-0421 and *z/OS DFSMShsm Storage Administration Reference,* SC35-0422, and visit:

```
http://www-1.ibm.com/servers/storage/software/sms/hsm/
```

4.55 DFSMShsm: Availability management

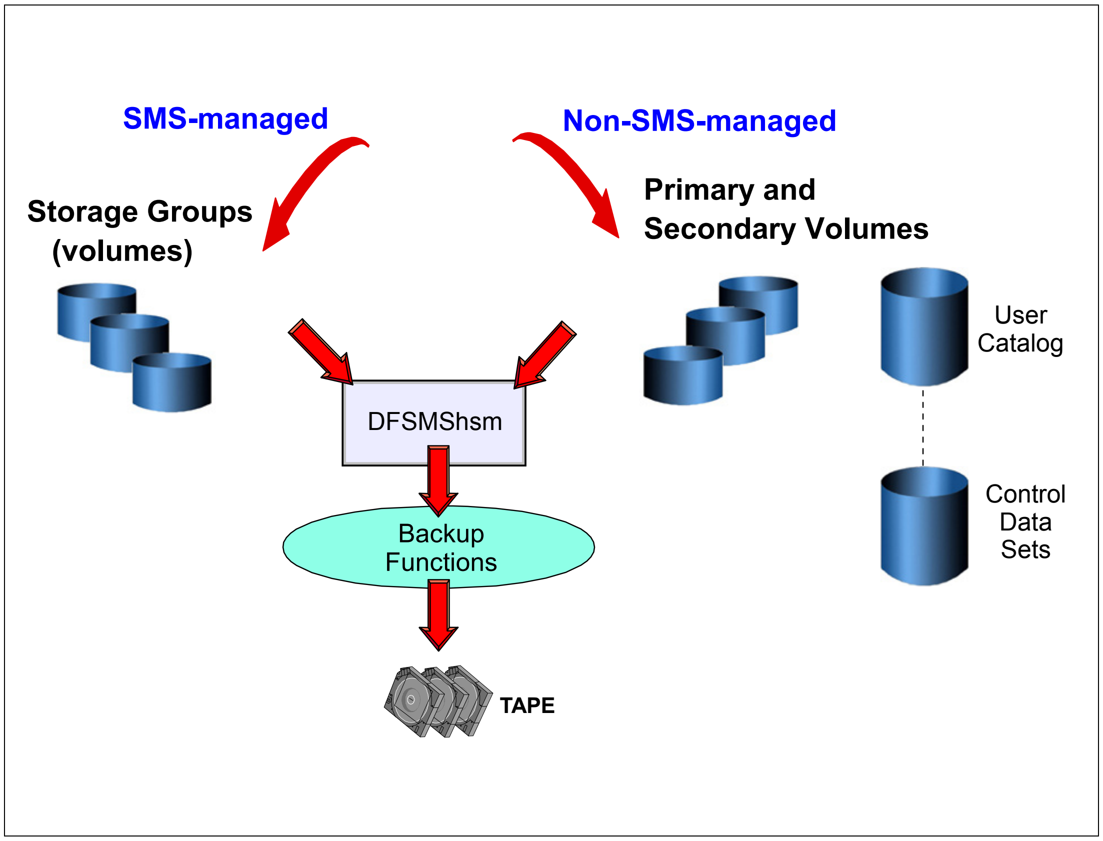


Figure 4-65 DFSMShsm availability management

Availability management

DFSMShsm backs up your data—automatically or by command—to ensure availability if accidental loss of the data sets or physical loss of volumes should occur. DFSMShsm also allows the storage administrator to copy backup and migration tapes, and to specify that copies be made in parallel with the original. You can store the copies on site as protection from media damage, or offsite as protection from site damage. DFSMShsm also provides disaster backup and recovery for user-defined groups of data sets (aggregates) so that you can restore critical applications at the same location or at an offsite location.

Note: You must also have DFSMSdss to use the DFSMShsm functions.

Availability management ensures that a recent copy of your DASD data set exists. The purpose of availability management is to ensure that lost or damaged data sets can be retrieved at the most current possible level. DFSMShsm uses DFSMSdss as a fast data mover for backups. Availability management automatically and periodically performs functions that:

1. Copy all the data sets on DASD volumes to tape volumes
2. Copy the changed data sets on DASD volumes (incremental backup) either to other DASD volumes or to tape volumes

DFSMShsm minimizes the space occupied by the data sets on the backup volume by using compression and stacking.

Availability management functions are:

- Aggregate backup and recovery (ABARS)
- Automatic physical full-volume dump
- Automatic incremental backup
- Automatic control data set backup
- Command dump and backup
- Command recovery
- Disaster backup
- Expiration of backup versions
- Fast replication backup and recovery

4.56 DFSMShsm: Space management

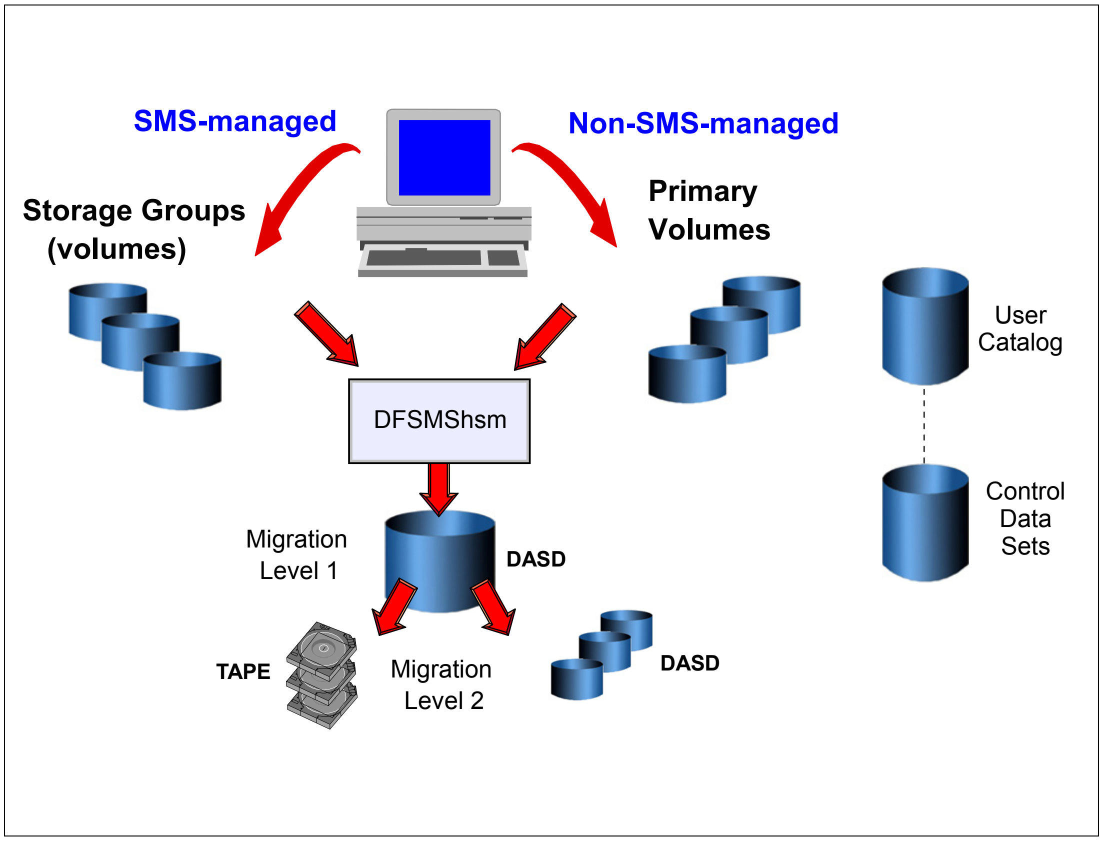


Figure 4-66 DFSMShsm space management

Space management

Space management is the function of DFSMShsm that allows you to keep DASD space available for users in order to meet the service level objectives for your system. The purpose of space management is to manage your DASD storage efficiently. To do this, space management automatically and periodically performs functions that:

1. Move low activity data sets (using DFSMSdss) from user-accessible volumes to DFSMShsm volumes
2. Reduce the space occupied by data on both the user-accessible volumes and the DFSMShsm volumes

DFSMShsm improves DASD space usage by keeping only active data on fast-access storage devices. It automatically frees space on user volumes by deleting eligible data sets, releasing overallocated space, and moving low-activity data to lower cost-per-byte devices, even if the job did not request tape.

Space management functions

The DFSMShsm space management functions are:

- Automatic primary space management of DFSMShsm-managed volumes, which includes:
  - Deletion of temporary data sets
  - Deletion of expired data sets

  - Release of unused, over-allocated space
  - Migration to DFSMShsm-owned migration level 1 (ML1) volumes (compressed)
- Automatic secondary space management of DFSMShsm-owned volumes, which includes:
  - Migration-level cleanup, including deletion of expired migrated data sets and some migration control data set (MCDS) records
  - Moving migration copies from migration level 1 (ML1) to migration level 2 (ML2) volumes
- Automatic interval migration, initiated when a DFSMShsm-managed volume exceeds a specified threshold
- Automatic recall of user data sets back to DASD volumes, when referenced by the application
- Space management by command
- Space-saving functions, which include:
  - Data compaction and data compression. Compaction provides space savings through fewer gaps and less control data. Compression provides a more compact way to store data.
  - Partitioned data set (PDS) free space compression.
  - Small data set packing (SDSP) data set facility, which allows small data sets to be packaged in just one physical track.
  - Data set reblocking.

It is possible to have more than one z/OS image sharing the same DFSMShsm policy. In this case one of the DFSMShsm images is the primary host and the others are secondary. The primary HSM host is identified by HOST= in the HSM startup and is responsible for:

- Hourly space checks
- During auto backup: CDS backup, backup of ML1 data sets to tape
- During auto dump: Expiration of dump copies and deletion of excess dump VTOC copy data sets
- During secondary space management (SSM): Cleanup of MCDS, migration volumes, and L1-to-L2 migration

If you are running your z/OS HSM images in sysplex (parallel or basic), you can use *secondary host promotion* to allow a secondary image to assume the primary image's tasks if the primary host fails. Secondary host promotion uses XCF status monitoring to execute the promotion. To indicate a system as a candidate, issue:

```
SETSYS PRIMARYHOST(YES)
```

and

```
SSM(YES)
```

4.57 DFSMShsm: Storage device hierarchy

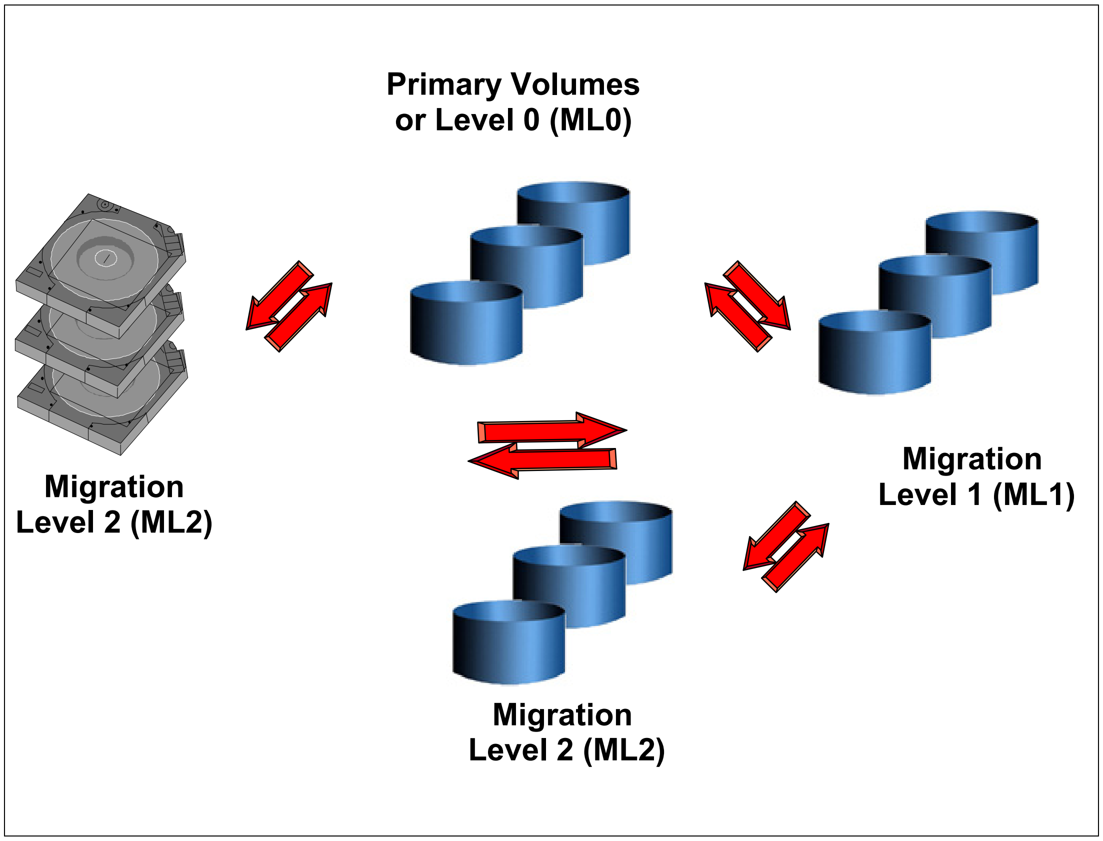


Figure 4-67 Storage device hierarchy

Storage device hierarchy

A storage device hierarchy consists of a group of storage devices that have different costs for storing data, different amounts of data stored, and different speeds of accessing the data.

DFSMShsm uses the following three-level storage device hierarchy for space management:

- Level 0: DFSMShsm-managed storage devices at the highest level of the hierarchy; these devices contain data directly accessible to your application.
- Level 1 and Level 2: Storage devices at the lower levels of the hierarchy; level 1 and level 2 contain data that DFSMShsm has compressed and optionally compacted into a format that you cannot use. Devices at this level provide lower cost per byte storage and usually slower response time. Usually L1 is in a cheaper DASD (or the same cost, but with the gain of compression) and L2 is on tape.

Note: If you have a DASD controller that compresses data, you can skip level 1 (ML1) migration because the data in L0 is already compacted/compressed.

4.58 DFSMShsm volume types

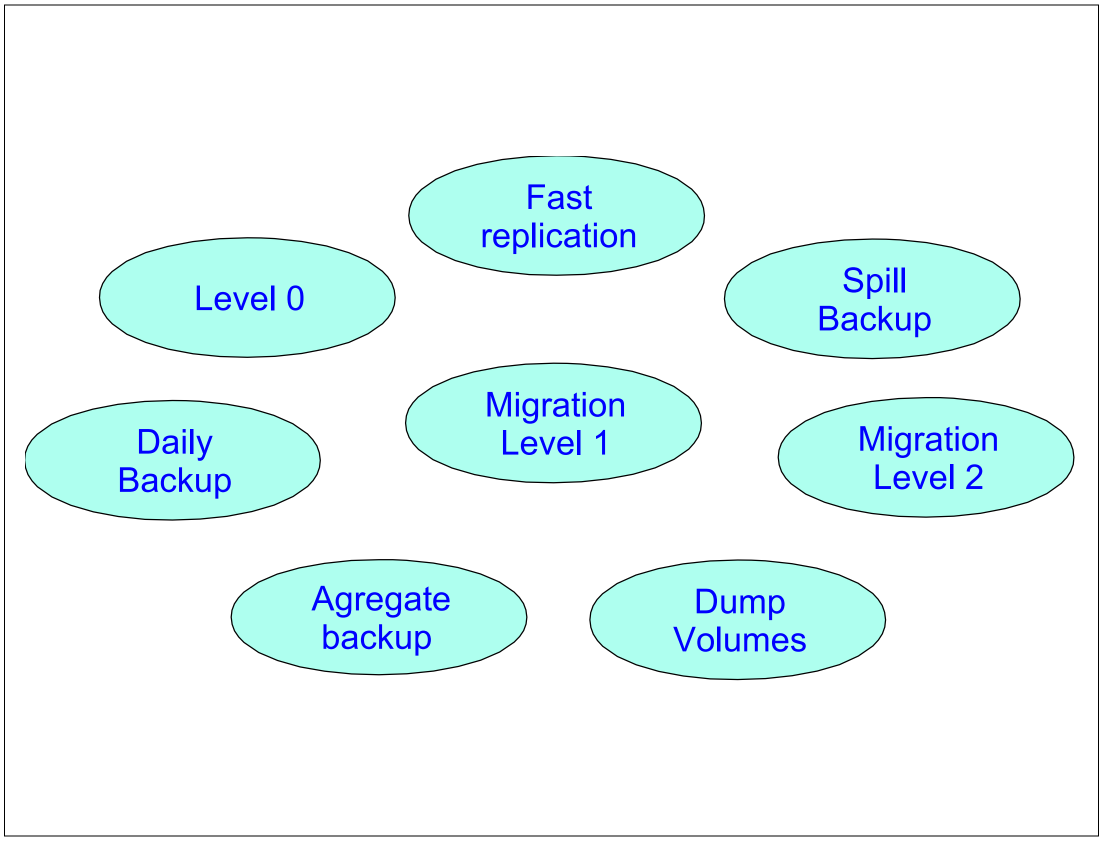


Figure 4-68 DFSMShsm volume types

DFSMShsm volume backup

Backing up an individual cataloged data set is performed in the same way as for SMS-managed data sets. However, to back up individual uncataloged data sets, issue the following commands:

```
BACKDS dsname UNIT(unittype) VOLUME(volser)
HBACKDS dsname UNIT(unittype) VOLUME(volser)
```

The **HBACKDS** form of the command can be used by either non-DFSMShsm-authorized or DFSMShsm-authorized users. The **BACKDS** form of the command can be used only by DFSMShsm-authorized users. The UNIT and VOLUME parameters are required because DFSMShsm cannot locate an uncataloged data set without being told where it is.

Volume types

DFSMShsm supports the following volume types:

- *Level 0 (L0) volumes* contain data sets that are directly accessible to you and the jobs you run. DFSMShsm-managed volumes are those L0 volumes that are managed by the DFSMShsm automatic functions. These volumes must be mounted and online when you refer to them with DFSMShsm commands.
- *Migration level 1 (ML1) volumes* are DFSMShsm-supported DASD on which DFSMShsm maintains your data in DFSMShsm format. These volumes are normally permanently mounted and online. They can be:

  - Volumes containing data sets that DFSMShsm migrated from L0 volumes.
  - Volumes containing backup versions created from a DFSMShsm **`BACKDS`** or **`HBACKDS`** command. Backup processing requires ML1 volumes to store incremental backup and dump VTOC copy data sets, and as intermediate storage for data sets that are backed up by data set command backup.

- *Migration level 2 (ML2) volumes* are DFSMShsm-supported tape or DASD on which DFSMShsm maintains your data in DFSMShsm format. These volumes are normally not mounted or online. They contain data sets migrated from ML1 volumes or L0 volumes.
- *Daily backup volumes* are DFSMShsm-supported tape or DASD on which DFSMShsm maintains your data in DFSMShsm format. These volumes are normally not mounted or online. They contain the most current backup versions of data sets copied from L0 volumes. These volumes may also contain earlier backup versions of these data sets.
- *Spill backup volumes* are DFSMShsm-supported tape or DASD on which DFSMShsm maintains your data sets in DFSMShsm format. These volumes are normally not mounted or online. They contain earlier backup versions of data sets, which were moved from DASD backup volumes.
- *Dump volumes* are DFSMShsm-supported tape. They contain image copies of volumes that are produced by the full volume dump function of DFSMSdss (write a copy of the entire allocated space of that volume), which is invoked by DFSMShsm.
- *Aggregate backup volumes* are DFSMShsm-supported tape. These volumes are normally not mounted or online. They contain copies of the data sets of a user-defined group of data sets, along with control information for those data sets. These data sets and their control information are stored as a group so that they can be recovered (if necessary) as an entity by an aggregate recovery process (ABARS).
- *Fast replication target volumes* are contained within SMS copy pool backup storage groups. They contain the fast replication backup copies of DFSMShsm-managed volumes. Since z/OS 1.8, fast replication is done with one single command.

New function with z/OS V1R7

With z/OS V1R7, the maximum number of data sets stored by DFSMShsm on tape is one million; previously, it was 330000.

Also in z/OS V1R7, a new command **`V SMS,VOLUME`** is introduced. It allows you to change the state of the DFSMShsm volumes without having to change and reactivate the SMS configuration using ISMF.

4.59 DFSMShsm: Automatic space management

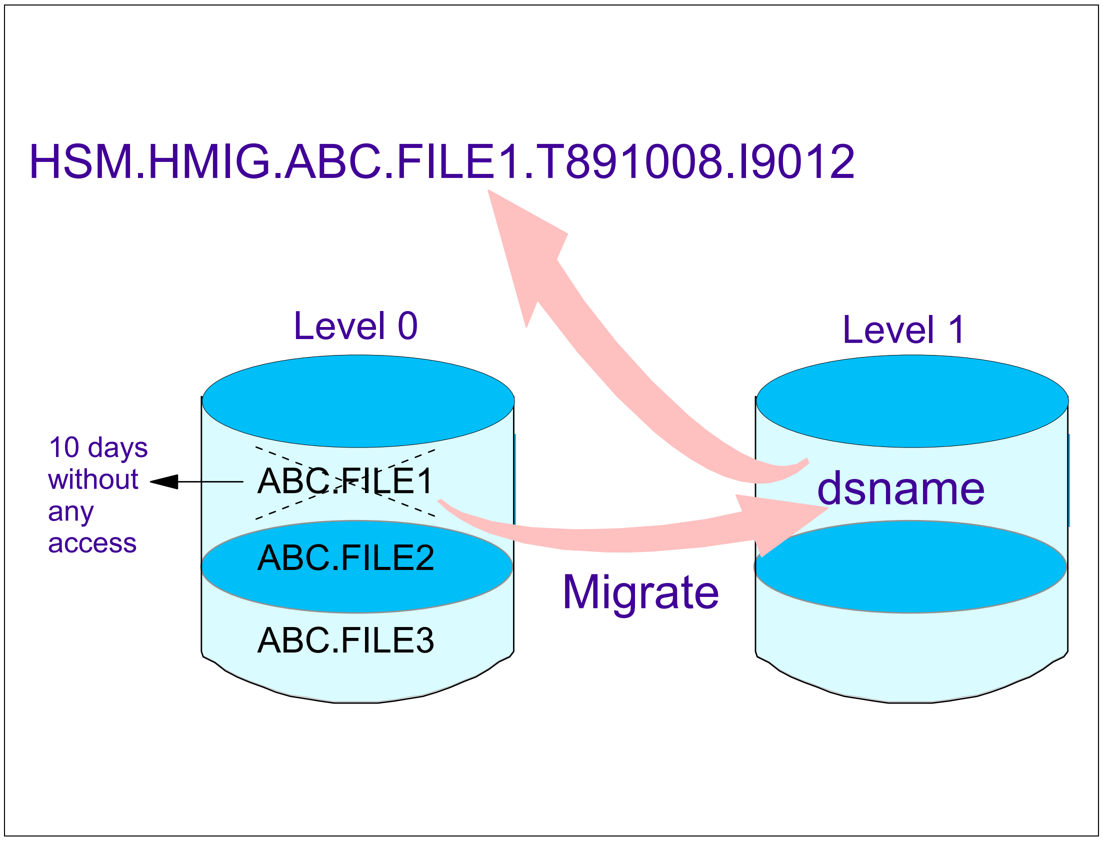


Figure 4-69 DFSMShsm automatic space management (migration)

Automatic space management (migration)

Automatic space management prepares DASD space for the addition of new data by freeing space on the DFSMShsm-managed volumes (L0) and DFSMShsm-owned volumes (ML1). The functions associated with automatic space management can be divided into two groups.

Automatic volume space management

Primary Invoked on a daily basis, it cleans L0 volumes by deleting expired and temporary data sets, releasing allocated and not used space (scratch). During automatic primary space management, DFSMShsm can process a maximum of 15 volume migration tasks concurrently. This activity consists of the deletion of temporary data sets, deletion of expired data sets, the release of unused and overallocated space, and migration. Each task processes its own separate user DASD volume. The storage administrator selects the maximum number of tasks that can run simultaneously, and specifies which days and the time of day the tasks are to be performed.

In z/OS V1R8 there is a specific task to scratch data sets. If after that the free space is still below a threshold, then it moves data sets (under control of management class) from L0 to ML1/ ML2 volumes.

Interval migration Executed each hour throughout the day, as needed for all storage groups. In interval migration, DFSMShsm performs a space check on each DFSMShsm volume being managed. A volume is considered eligible for interval migration based on the AUTOMIGRATE and THRESHOLD settings of its SMS storage group.

Automatic interval migration is an option that invokes migration when DFSMShsm-managed volumes become full during high activity periods. If the storage administrator chooses this option, DFSMShsm automatically checks the level of occupancy of all DFSMShsm-managed volumes periodically. If the level of occupancy for any volume exceeds a given threshold, DFSMShsm automatically performs a subset of the space management functions on the volume. The threshold you select should be one that would be exceeded only when your installation's activity exceeds its usual peak. For those volumes requiring interval migration, DFSMShsm can process up to 15 volume migration tasks concurrently.

During automatic interval migration on a volume, the expired data sets are deleted, then the largest eligible data sets are moved first so that the level of occupancy threshold can be reached sooner. Data sets are not migrated from ML1 to ML2 volumes during interval migration.

Automatic secondary space management

This deletes expired data sets from ML1/ML2, then moves data sets (under control of the management class) from ML1 to ML2 volumes. It should complete before automatic primary space management so that the ML1 volumes will not run out of space. Since z/OS 1.6 there is the possibility of multiple secondary space management (SSM) tasks.

4.60 DFSMShsm: Recall processing

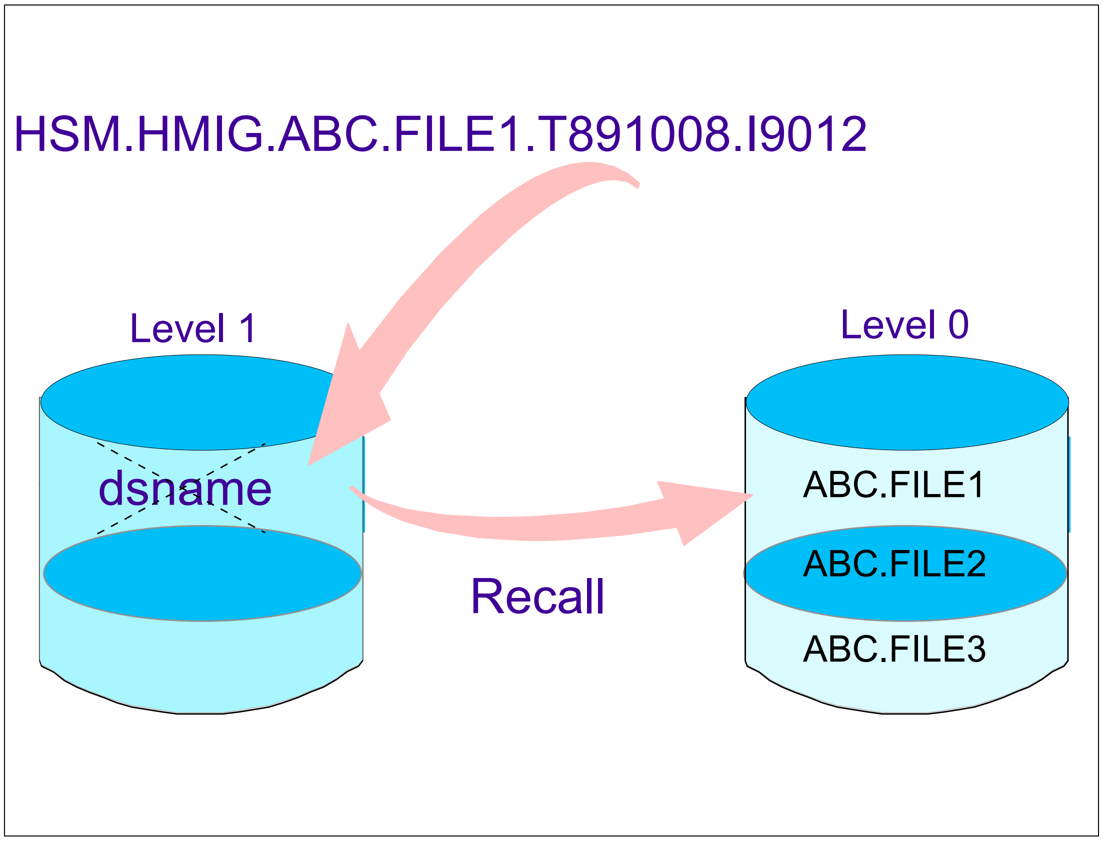


Figure 4-70 Recall processing

Automatic recall

Using an automatic recall process returns a migrated data set from an ML1 or ML2 volume to a DFSMShsm-managed volume. When a user refers to the data set, DFSMShsm reads the system catalog for the volume serial number. If the volume serial number is MIGRAT, DFSMShsm finds the migrated data set, recalls it to a DFSMShsm-managed volume, and updates the catalog. The result of the recall process is a data set that resides on a user volume in a user readable format. The recall can also be requested by a DFSMShsm command. *Automatic recall* returns your migrated data set to a DFSMShsm-managed volume when you refer to it. The catalog is updated accordingly with the real volser.

Recall returns a migrated data set to a user L0 volume. The recall is transparent and the application does not need to know that it happened or where the migrated data set resides. To provide applications with quick access to their migrated data sets, DFSMShsm allows up to 15 concurrent recall tasks. RMF monitor III shows delays caused by the recall operation.

The MVS allocation routine discovers that the data set is migrated when, while accessing the catalog, it finds the word MIGRAT instead of the volser.

Command recall

Command recall returns your migrated data set to a user volume when you enter the **HRECALL** DFSMShsm command through an ISMF panel or by directly keying in the command.

ACS routines

For both automatic and command recall, DFSMShsm working with SMS invokes the automatic class selection (ACS) routines. Data sets that were not SMS-managed at the time they were migrated may be recalled as SMS-managed data sets. The ACS routines determine whether the data sets should be recalled as SMS-managed, and if so, the routines select the classes and storage groups in which the data sets will reside. The system chooses the appropriate volume for the data sets.

DFSMShsm working without SMS returns a migrated data set to a DFSMShsm-managed non-SMS level 0 volume with the most free space.

4.61 Removable media manager (DFSMSrmm)

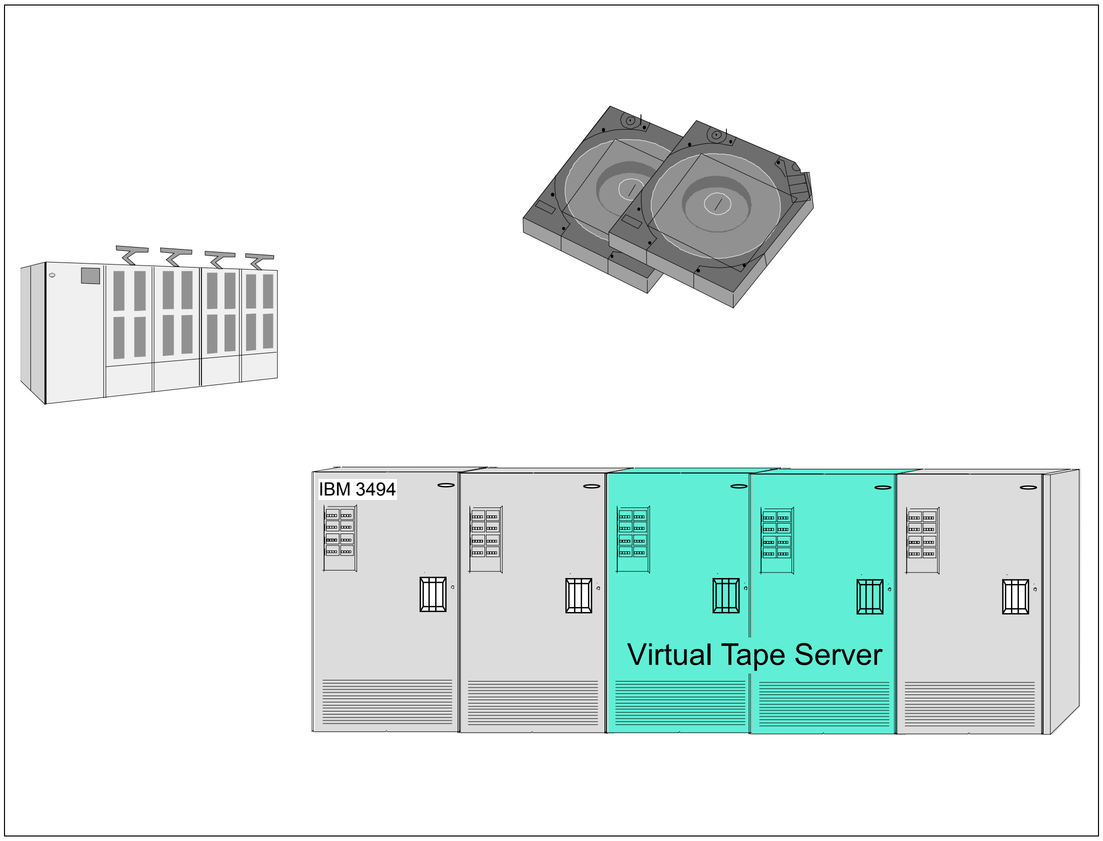


Figure 4-71 DFSMSrmm

DFSMSrmm

In your enterprise, you store and manage your removable media in several types of media libraries. For example, in addition to your traditional tape library (a room with tapes, shelves, and drives), you might have several automated and manual tape libraries. You probably also have both onsite libraries and offsite storage locations, also known as *vaults* or *stores*.

With the DFSMSrmm functional component of DFSMS, you can manage your removable media as one enterprise-wide library (single image) across systems. Because of the need for global control information, these systems must have accessibility to some shared DASD volumes. DFSMSrmm manages your installation's tape volumes and the data sets on those volumes. DFSMSrmm also manages the shelves where volumes reside in all locations except in automated tape library data servers.

DFSMSrmm manages all tape media (such as cartridge system tapes and 3420 reels), as well as other removable media you define to it. For example, DFSMSrmm can record the shelf location for optical disks and track their vital record status; however, it does not manage the objects on optical disks.

Library management

DFSMSrmm can manage the following devices:

- ► A removable media library, which incorporates all other libraries, such as:
  - – System-managed manual tape libraries

  - System-managed automated tape libraries
- Non-system-managed or traditional tape libraries, including automated libraries such as a library under Basic Tape Library Support (BTLS) control.

Examples of automated tape libraries include IBM TotalStorage Enterprise Automated Tape Library (3494) and IBM TotalStorage Virtual Tape Servers (VTS).

Shelf management

DFSMSrmm groups information about removable media by shelves into a central online inventory, and keeps track of the volumes residing on those shelves. DFSMSrmm can manage the shelf space that you define in your removable media library and in your storage locations.

Volume management

DFSMSrmm manages the movement and retention of tape volumes throughout their life cycle.

Data set management

DFSMSrmm records information about the data sets on tape volumes. DFSMSrmm uses the data set information to validate volumes and to control the retention and movement of those data sets.

For more information about DFSMSrmm, refer to *z/OS DFSMSrmm Guide and Reference,* SC26-7404 and *z/OS DFSMSrmm Implementation and Customization Guide,* SC26-7405, and visit:

```
http://www-1.ibm.com/servers/storage/software/sms/rmm/
```

4.62 Libraries and locations

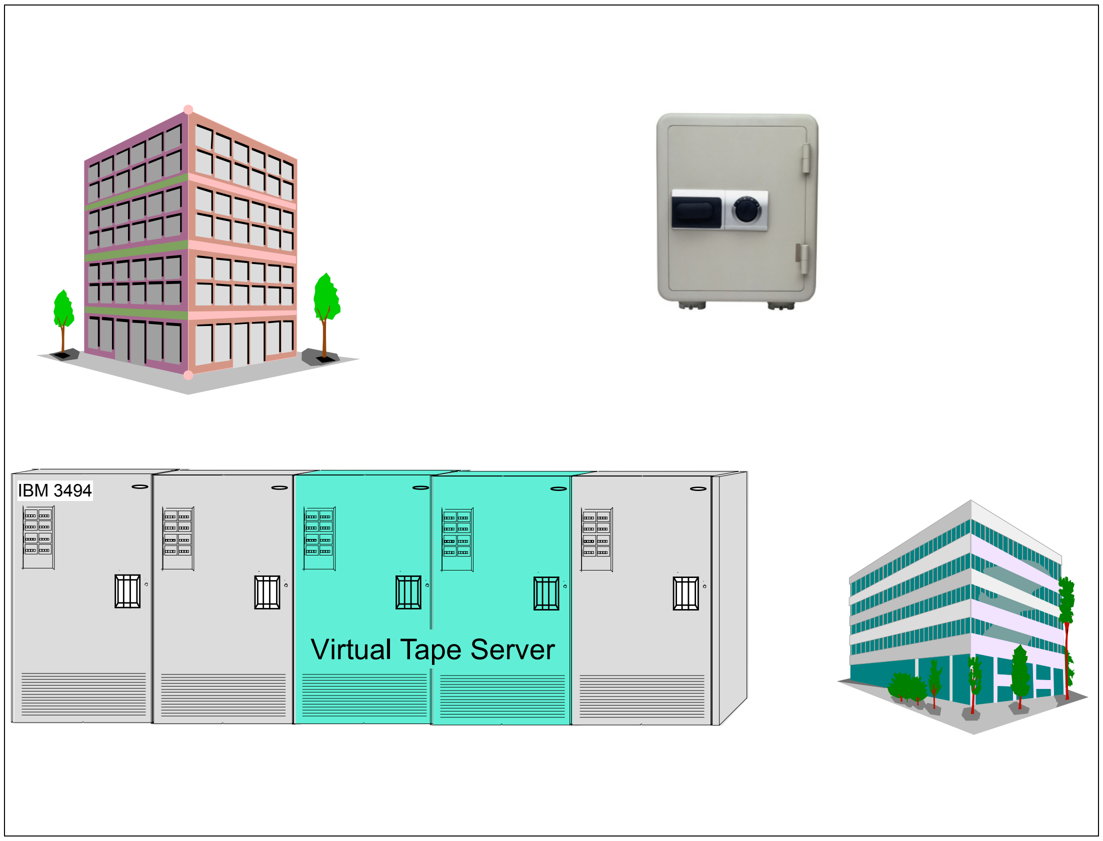


Figure 4-72 Libraries and locations

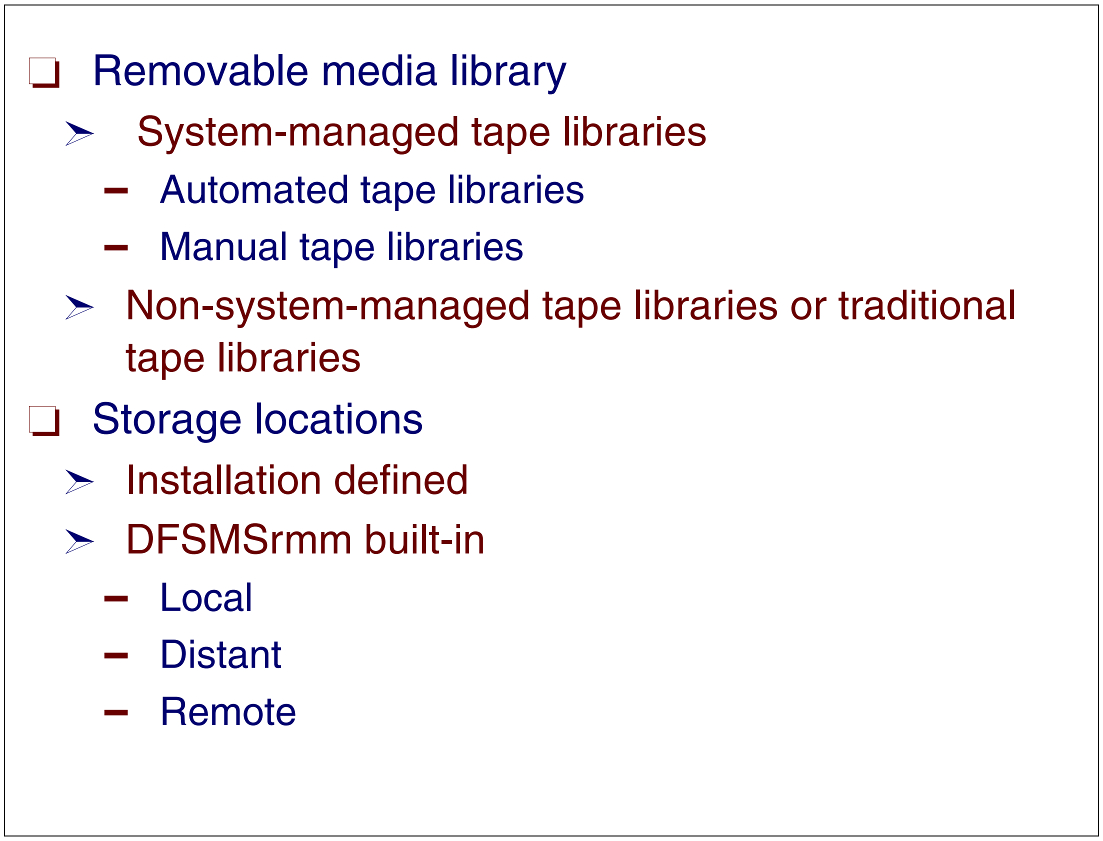
Libraries and locations

You decide where to store your removable media based on how often the media is accessed and for what purpose it is retained. For example, you might keep volumes that are frequently accessed in an automated tape library data server, and you probably use at least one storage location to retain volumes for disaster recovery and audit purposes. You might also have locations where volumes are sent for further processing, such as other data centers within your company or those of your customers and vendors.

DFSMSrmm automatically records information about data sets on tape volumes so that you can manage the data sets and volumes more efficiently. When all the data sets on a volume have expired, the volume can be reclaimed and reused. You can optionally move volumes that are to be retained to another location.

DFSMSrmm helps you manage your tape volumes and shelves at your primary site and storage locations by recording information in a DFSMSrmm control data set.

4.63 What DFSMSrmm can manage



- Removable media library
  - System-managed tape libraries
    - Automated tape libraries
    - Manual tape libraries
  - Non-system-managed tape libraries or traditional tape libraries
- Storage locations
  - Installation defined
  - DFSMSrmm built-in
    - Local
    - Distant
    - Remote

Figure 4-73 What DFSMSrmm can manage

What DFSMSrmm can manage

In this section we cover libraries and storage locations that can be managed by DFSMSrmm.

Removable media library

A removable media library contains all the tape and optical volumes that are available for immediate use, including the shelves where they reside. A removable media library usually includes other libraries:

- *System-managed libraries*, such as automated or manual tape library data servers
- *Non-system-managed libraries*, containing the volumes, shelves, and drives not in an automated or a manual tape library data server

In the removable media library, you store your volumes in “shelves,” where each volume occupies a single shelf location. This shelf location is referred to as a *rack number* in the DFSMSrmm TSO subcommands and ISPF dialog. A rack number matches the volume’s external label. DFSMSrmm uses the external volume serial number to assign a rack number when adding a volume, unless you specify otherwise. The format of the volume serial you define to DFSMSrmm must be one to six alphanumeric characters. The rack number must be six alphanumeric or national characters.

System-managed tape library

A system-managed tape library is a collection of tape volumes and tape devices defined in the tape configuration database. The tape configuration database is an integrated catalog facility user catalog marked as a *volume catalog* (VOLCAT) containing tape volumes and tape library records. A system-managed tape library can be either automated or manual:

- An *automated tape library* is a device consisting of robotic components, cartridge storage areas (or shelves), tape subsystems, and controlling hardware and software, together with the set of tape volumes that reside in the library and can be mounted on the library tape drives. The IBM automated tape libraries are the automated IBM 3494 and IBM 3495 Library Dataservers.
- A *manual tape library* is a set of tape drives and the set of system-managed volumes the operator can mount on those drives. The manual tape library provides more flexibility, enabling you to use various tape volumes in a given manual tape library. Unlike the automated tape library, the manual tape library does not use the library manager. With the manual tape library, a human operator responds to mount messages that are generated by the host and displayed on a console. This manual tape library implementation completely replaces the IBM 3495-M10 implementation. IBM no longer supports the 3495-M10.

You can have several automated tape libraries or manual tape libraries. You use an installation-defined library name to define each automated tape library or manual tape library to the system. DFSMSrmm treats each system-managed tape library as a separate location or destination.

Since z/OS 1.6, a new EDGRMMxx parmlib member **`OPTION`** command, together with **`VLPOOL`** command, allows better support for the client/server environment.

z/OS 1.8 DFSMSrmm introduces an option to provide tape data set authorization independent of the RACF TAPVOL and TAPEDSN. This option allows you to use RACF generic DATASET profiles for both DASD and tape data sets.

Non-system-managed tape library

A non-system-managed tape library consists of all the volumes, shelves, and drives not in an automated tape library or manual tape library. You might know this library as the traditional tape library that is not system-managed. DFSMSrmm provides complete tape management functions for the volumes and shelves in this traditional tape library. Volumes in a non-system-managed library are defined by DFSMSrmm as being “shelf-resident”.

All tape media and drives supported by z/OS are supported in this environment. Using DFSMSrmm, you can fully manage all types of tapes in a non-system-managed tape library, including 3420 reels, 3480, 3490, and 3590 cartridge system tapes.

Storage location

Storage locations are not part of the removable media library because the volumes in storage locations are not generally available for immediate use. A *storage location* is comprised of shelf locations that you define to DFSMSrmm. A *shelf location* in a storage location is identified by a bin number. Storage locations are typically used to store removable media that are kept for disaster recovery or vital records. DFSMSrmm manages two types of storage locations: installation-defined storage locations and DFSMSrmm built-in storage locations.

You can define an unlimited number of installation-defined storage locations, using any eight-character name for each storage location. Within the installation-defined storage location, you can define the type or shape of the media in the location. You can also define the bin numbers that DFSMSrmm assigns to the shelf locations in the storage location. You can request DFSMSrmm shelf-management when you want DFSMSrmm to assign a specific shelf location to a volume in the location.

You can also use the DFSMSrmm built-in storage locations: LOCAL, DISTANT, and REMOTE. Although the names of these locations imply their purpose, they do not mandate their actual location. All volumes can be in the same or separate physical locations. For example, an installation could have the LOCAL storage location on-site, as a vault in the computer room, the DISTANT storage location could be a vault in an adjacent building, and the REMOTE storage location could be a secure facility across town or in another state. DFSMSrmm provides shelf-management for storage locations so that storage locations can be managed at the shelf location level.

4.64 Managing libraries and storage locations

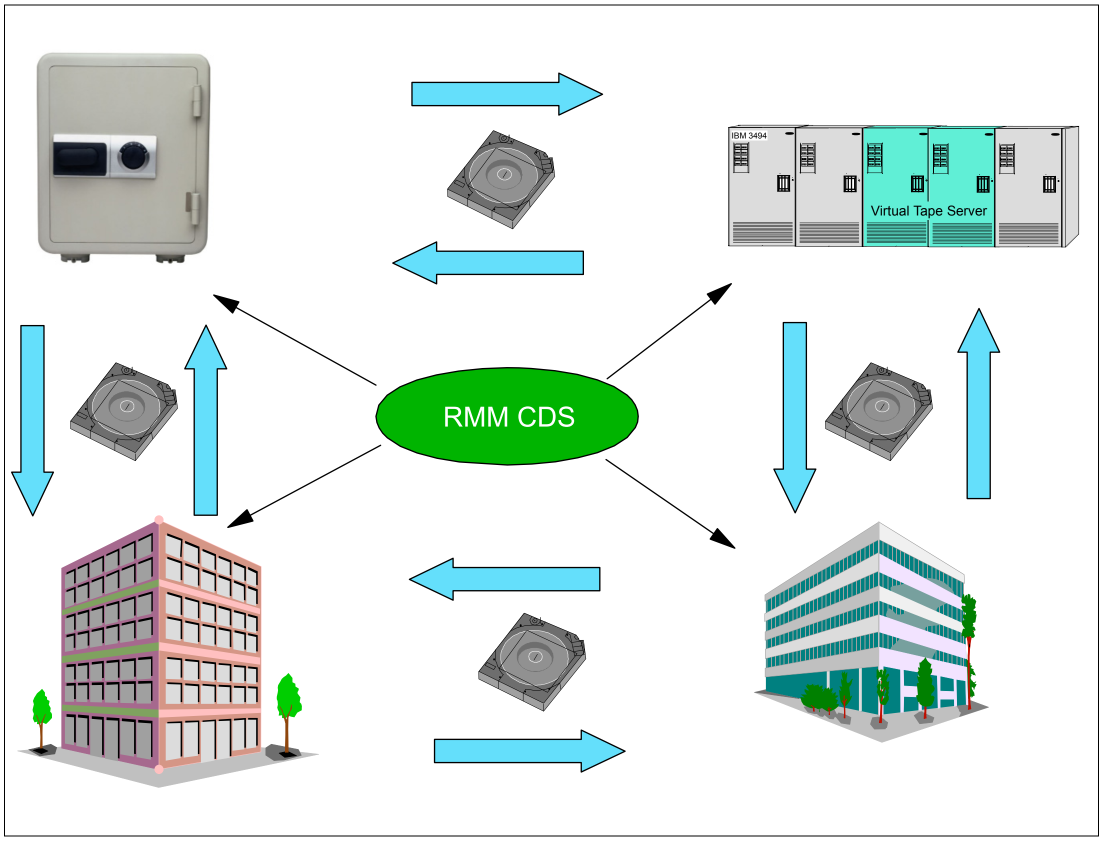


Figure 4-74 DFSMSrmm: managing libraries and storage locations

Managing libraries and storage locations

DFSMSrmm records the complete inventory of the removable media library and storage locations in the DFSMSrmm control data set, which is a VSAM key-sequenced data set. In the control data set, DFSMSrmm records all changes made to the inventory (such as adding or deleting volumes), and also keeps track of all movement between libraries and storage locations. DFSMSrmm manages the movement of volumes among all library types and storage locations. This lets you control where a volume—and hence, a data set—resides, and how long it is retained.

DFSMSrmm helps you manage the movement of your volumes and retention of your data over their full life, from initial use to the time they are retired from service. Among the functions DFSMSrmm performs for you are:

- Automatically initializing and erasing volumes
- Recording information about volumes and data sets as they are used
- Expiration processing
- Identifying volumes with high error levels that require replacement

To make full use of all of the DFSMSrmm functions, you specify installation setup options and define retention and movement policies. DFSMSrmm provides you with utilities to implement the policies you define. Since z/OS 1.7, we have DFSMSrmm enterprise enablement that allows high-level languages to issue DFSMSrmm commands through Web services.

z/OS V1R8 enhancements

DFSMSrmm helps you manage the shelves in your tape library and storage locations, simplifying the tasks of your tape librarian. When you define a new volume in your library, you can request that DFSMSrmm shelf-manage the volume by assigning the volume a place on the shelf. You also have the option to request a specific place for the volume. Your shelves are easier to use when DFSMSrmm manages them in pools. Pools allow you to divide your shelves into logical groups where you can store volumes. For example, you can have a different pool for each system that your installation uses. You can then store the volumes for each system together in the same pool.

You can define shelf space in storage locations. When you move volumes to a storage location where you have defined shelf space, DFSMSrmm checks for available shelf space and then assigns each volume a place on the shelf if you request it. You can also set up DFSMSrmm to reuse shelf space in storage locations.

5

System-managed storage

As your business expands, so does your need for storage to hold your applications and data, and the cost of managing that storage. Storage cost includes more than the price of the hardware, with the highest cost being the people needed to perform storage management tasks. If your business requires transaction systems, the batch window can also be a high cost. Additionally, you must pay for people to install, monitor, and operate your storage hardware devices, for electrical power to keep each piece of storage hardware cool and running, and for floor space to house the hardware. Removable media, such as optical and tape storage, cost less per gigabyte (GB) than online storage, but require additional time and resources to locate, retrieve, and mount.

To allow your business to grow efficiently and profitably, you need to find ways to control the growth of your information systems and use your current storage more effectively.

With these goals in mind, in this chapter we present:

- The z/OS storage-managed environment
- Benefits of a system-managed environment
- An overview of how SMS manages a storage environment based on installation policies
- How to set up a minimal SMS configuration and activate a DFSMS subsystem
- How to manage data using a minimal SMS configuration
- How to use Interactive Storage Management Facility (ISMF), an interface for defining and maintaining storage management policies

5.1 Storage management

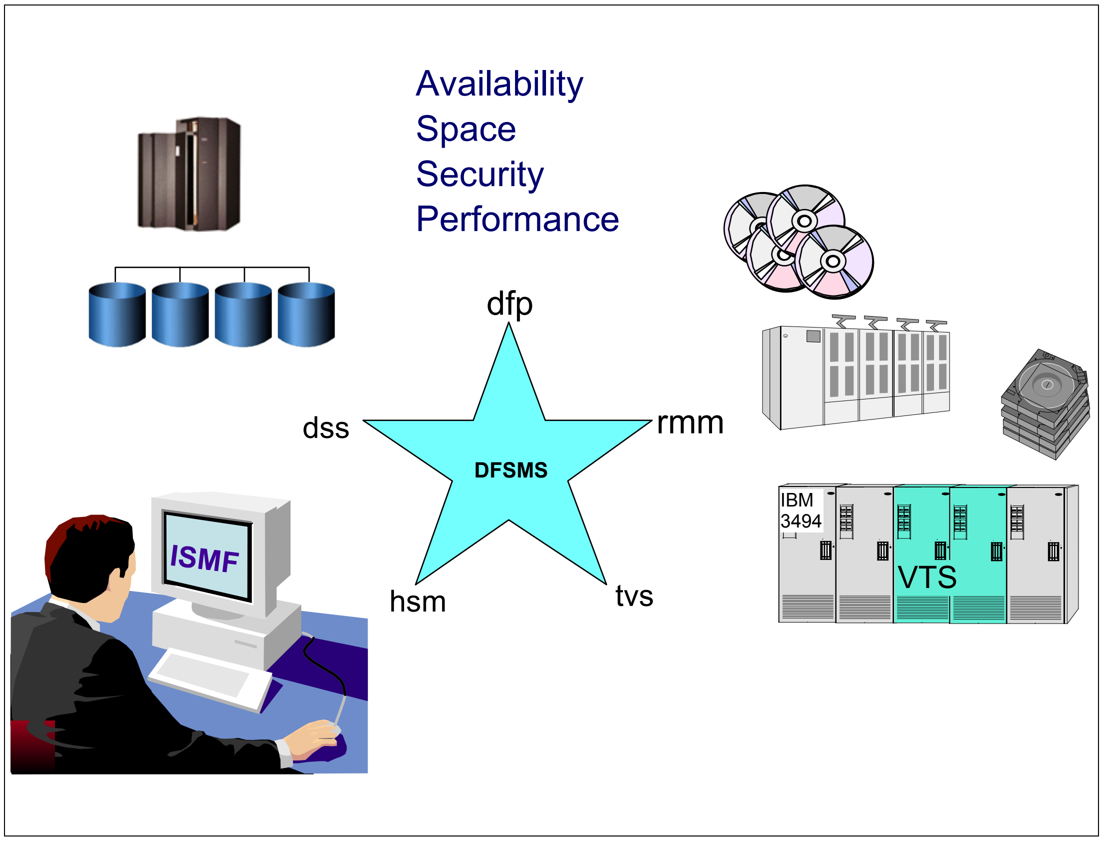


Figure 5-1 Managing storage with DFSMS

Storage management

Storage management involves data set allocation, placement, monitoring, migration, backup, recall, recovery, and deletion. These activities can be done either manually or by using automated processes.

Managing storage with DFSMS

The *Data Facility Storage Management Subsystem* (DFSMS) comprises the base z/OS operating system and performs the essential data, storage, program, and device management functions of the system. DFSMS is the central component of both system-managed and non-system-managed storage environments.

The DFSMS software product, together with hardware products and installation-specific requirements for data and resource management, comprises the key to system-managed storage in a z/OS environment.

The heart of DFSMS is the *Storage Management Subsystem* (SMS). Using SMS, the storage administrator defines policies that automate the management of storage and hardware devices. These policies describe data allocation characteristics, performance and availability goals, backup and retention requirements, and storage requirements for the system. SMS governs these policies for the system and the *Interactive Storage Management Facility* (ISMF) provides the user interface for defining and maintaining the policies.

5.2 DFSMS and DFSMS environment

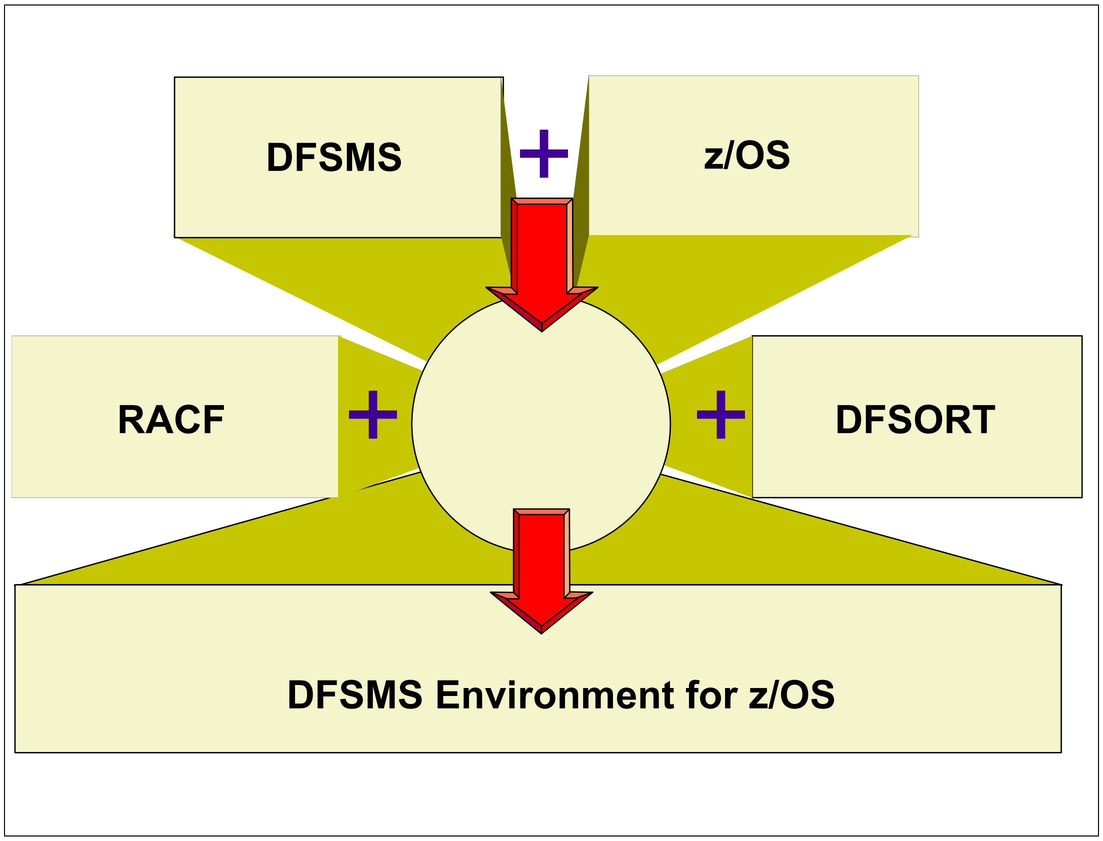


Figure 5-2 SMS environment

DFSMS functional components

DFSMS is a set of products, and one of these products, DSFMSdfp, is mandatory for running z/OS. DFSMS comprises the base z/OS operating system, where DFSMS performs the essential data, storage, program, and device management functions of the system. DFSMS is the central component of both system-managed and non-system-managed storage environments.

DFSMS environment

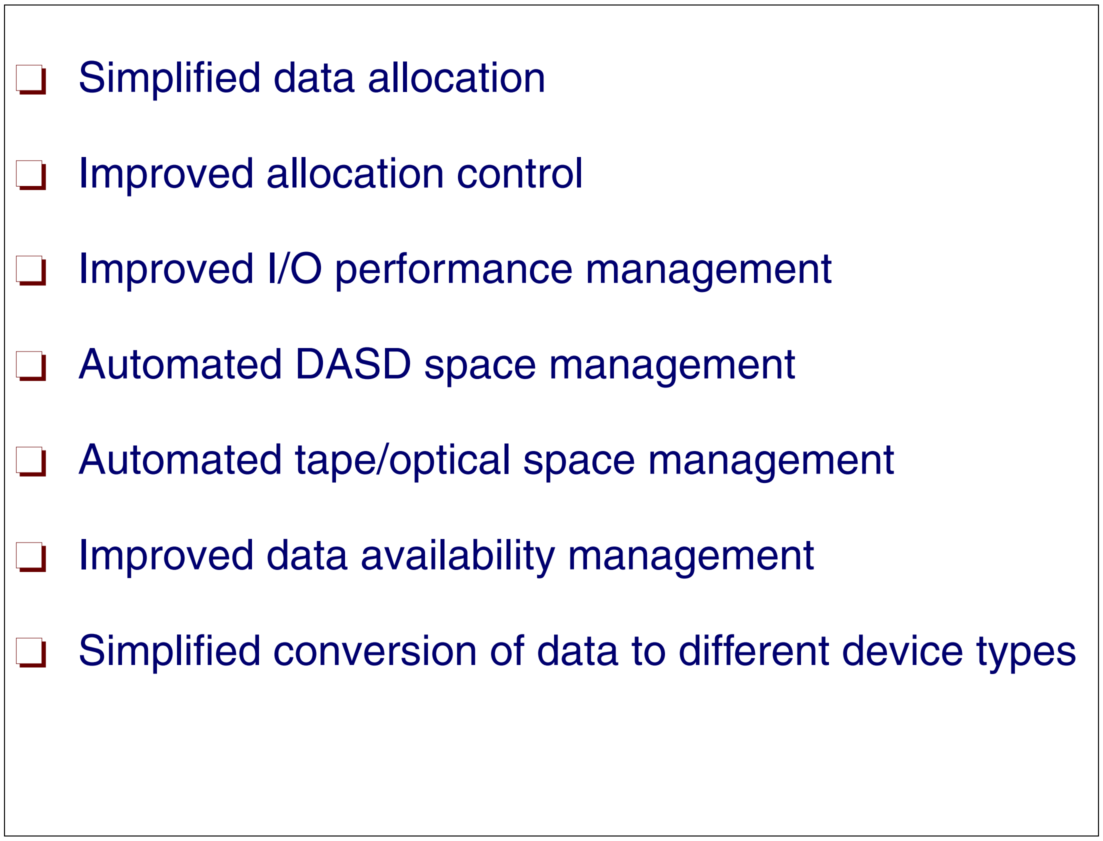
The DFSMS environment consists of a set of hardware and IBM software products which together provide a system-managed storage solution for z/OS installations.

DFSMS uses a set of constructs, user interfaces, and routines (using the DFSMS products) that allow the storage administrator to better manage the storage system. The core logic of DFSMS, such as the Automatic Class Selection (ACS) routines, ISMF code, and constructs, is located in DFSMSdfp. DFSMShsm and DFSMSdss are involved in the management class construct.

In this environment, the *Resource Access Control Facility* (RACF) and *Data Facility Sort* (DFSORT) products complement the functions of the base operating system. RACF provides resource security functions, and DFSORT adds the capability for faster and more efficient sorting, merging, copying, reporting, and analyzing of business information.

The DFSMS environment is also called the *SMS environment*.

5.3 Goals and benefits of system-managed storage



- ❑ Simplified data allocation
- ❑ Improved allocation control
- ❑ Improved I/O performance management
- ❑ Automated DASD space management
- ❑ Automated tape/optical space management
- ❑ Improved data availability management
- ❑ Simplified conversion of data to different device types

Figure 5-3 Benefits of system-managed storage

Benefits of system-managed storage

With the Storage Management Subsystem (SMS), you can define performance goals and data availability requirements, create model data definitions for typical data sets, and automate data backup. Based on installation policies, SMS can automatically assign those services and data definition attributes to data sets when they are created. IBM storage management-related products determine data placement, manage data backup, control space usage, and provide data security.

Goals of system-managed storage

The goals of system-managed storage are:

- ► To improve the use of the storage media (for example, by reducing out-of-space abends and providing a way to set a free-space requirement).
- ► To reduce the labor involved in storage management by centralizing control, automating tasks, and providing interactive controls for storage administrators.
- ► To reduce the user's need to be concerned with the physical details, performance, space, and device management. Users can focus on using data, instead of on managing data.

Simplified data allocation

System-managed storage enables users to simplify their data allocations. For example, without using the Storage Management Subsystem, a z/OS user would have to specify the unit and volume on which the system should allocate the data set. The user would also have to calculate the amount of space required for the data set in terms of tracks or cylinders. This means the user has to know the track size of the device that will contain the data set. With system-managed storage, users can let the system select the specific unit and volume for the allocation. They can also specify size requirements in terms of megabytes or kilobytes. This means the user does not need to know anything about the physical characteristics of the devices in the installation.

Improved allocation control

System-managed storage enables you to set a requirement for free space across a set of direct access storage device (DASD) volumes. You can then provide adequate free space to avoid out-of-space abends. The system automatically places data on a volume containing adequate free space. DFSMS offers relief to avoid out-of-space conditions. You can also set a threshold for scratch tape volumes in tape libraries, to ensure enough cartridges are available in the tape library for scratch mounts.

Improved Input/Output (I/O) performance management

System-managed storage enables you to improve DASD I/O performance across the installation. At the same time, it reduces the need for manual tuning by defining performance goals for each class of data. You can use cache statistics recorded in System Management Facility (SMF) records to help evaluate performance. You can also improve sequential performance by using *extended* sequential data sets. The DFSMS environment makes the most effective use of the caching abilities of storage controllers.

Automated DASD space management

System-managed storage enables you to automatically reclaim space that is allocated to old and unused data sets or objects. You can define policies that determine how long an unused data set or object will be allowed to reside on primary storage (storage devices used for your active data). You can have the system remove obsolete data by migrating the data to other DASD, tape, or optical volumes, or you can have the system delete the data. You can also release allocated but unused space that is assigned to new and active data sets.

Automated tape space management

Mount Management System-managed storage lets you fully use the capacity of your tape cartridges and automate tape mounts. Using *tape mount management* (TMM) methodology, DFSMShsm can fill tapes to their capacity. With 3490E, 3590, 3591, and 3592 tape devices, Enhanced Capacity Cartridge System Tape, recording modes such as 384-track and EFMT1, and the improved data recording capability, you can increase the amount of data that can be written on a single tape cartridge.

Tape System-managed storage lets you exploit the device technology of new devices without having to change the JCL UNIT parameter. In a multi-library environment, you can select the drive based on the library where the cartridge or volume resides. You can use the IBM TotalStorage Enterprise Automated Tape Library (3494 or 3495) to automatically mount tape volumes and manage the inventory in an automated tape library. Similar functionality is available in a system-managed manual tape library. If you are not using SMS for tape management, you can still access the IBM TotalStorage Enterprise Automated Tape Library (3494 or 3495) using Basic Tape Library Storage (BTLS) software.

Automated optical space management

System-managed storage enables you to fully use the capacity of your optical cartridges and to automate optical mounts. Using a 3995 Optical Library Dataserver, you can automatically mount optical volumes and manage the inventory in an automated optical library.

Improved data availability management

System-managed storage enables you to provide different backup requirements to data residing on the same DASD volume. Thus, you do not have to treat all data on a single volume the same way.

You can use DFSMShsm to automatically back up your different types of data sets and use point-in-time copy to maintain access to critical data sets while they are being backed up. Concurrent copy, virtual concurrent copy, SnapShot, and FlashCopy, along with backup-while-open, have an added advantage in that they avoid invalidating a backup of a CICS VSAM KSDS due to a control area or control interval split.

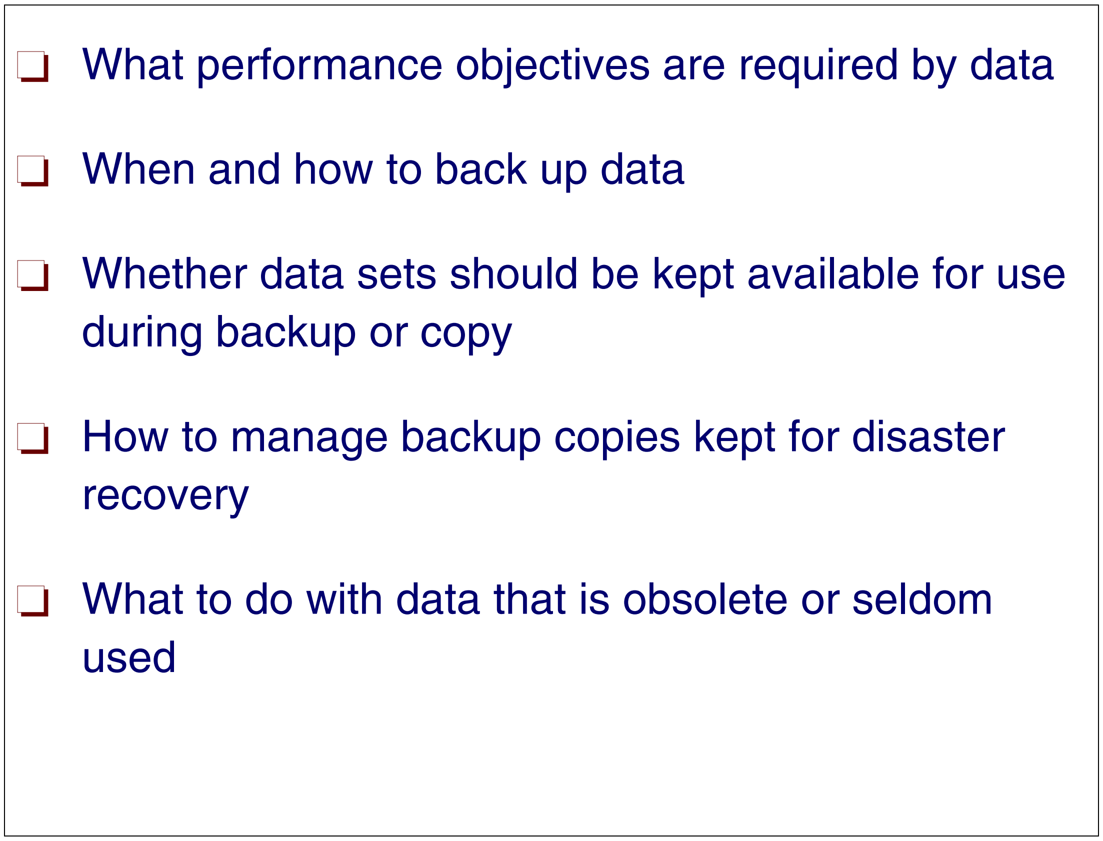
You can also create a logical group of data sets, so that the group is backed up at the same time to allow recovery of the application defined by the group. This is done with the aggregate backup and recovery support (ABARS) provided by DFSMShsm.

Simplified conversion of data to different device types

System-managed storage enables you to move data to new volumes without requiring users to update their job control language (JCL). Because users in a DFSMS environment do not need to specify the unit and volume which contains their data, it does not matter to them if their data resides on a specific volume or device type. This allows you to easily replace old devices with new ones.

You can also use system-determined block sizes to automatically reblock physical sequential and partitioned data sets that can be reblocked.

5.4 Service level objectives



- ❑ What performance objectives are required by data
- ❑ When and how to back up data
- ❑ Whether data sets should be kept available for use during backup or copy
- ❑ How to manage backup copies kept for disaster recovery
- ❑ What to do with data that is obsolete or seldom used

Figure 5-4 Service level objectives

Service level objectives

To allow your business to grow efficiently and profitably, you want to find ways to control the growth of your information systems and use your current storage more effectively.

In an SMS-managed storage environment, your enterprise establishes centralized policies for how to use your hardware resources. These policies balance your available resources with your users' requirements for data availability, performance, space, and security.

The policies defined in your installation represent decisions about your resources, such as:

- What performance objectives are required by the applications accessing the data

  Based on these objectives, you can try to better exploit cache data striping. By tracking data set I/O activities, you can make better decisions about data set caching policies and improve overall system performance. For object data, you can track transaction activities to monitor and improve OAM's performance.

- When and how to back up data - incremental or total

  Determine the backup frequency, the number of backup versions, and the retention period by consulting user group representatives. Be sure to consider whether certain data backups need to be synchronized. For example, if the output data from application A is used as input for application B, you must coordinate the backups of both applications to prevent logical errors in the data when they are recovered.

- Whether data sets should be kept available for use during backup or copy

You can store backup data sets on DASD or tape (this does not apply to objects). Your choice depends on how fast the data needs to be recovered, media cost, operator cost, floor space, power requirements, air conditioning, the size of the data sets, and whether you want the data sets to be portable.

- How to manage backup copies kept for disaster recovery - locally or in a vault

  Related data sets should be backed up in aggregated tapes. Each application should have its own, self-contained aggregate of data sets. If certain data sets are shared by two or more applications, you might want to ensure application independence for disaster recovery by backing up each application that shares the data. This is especially important for shared data in a distributed environment.

- What to do with data that is obsolete or seldom used

  Data is obsolete when it has exceeded its expiration dates and is no longer needed. To select obsolete data for deletion using DFSMSdss, issue the DUMP command and the DELETE parameter, and force OUTDDNAME to DUMMY.

The purpose of a backup plan is to ensure the prompt and complete recovery of data. A well-documented plan identifies data that requires backup, the levels required, responsibilities for backing up the data, and methods to be used.

5.5 Implementing SMS policies

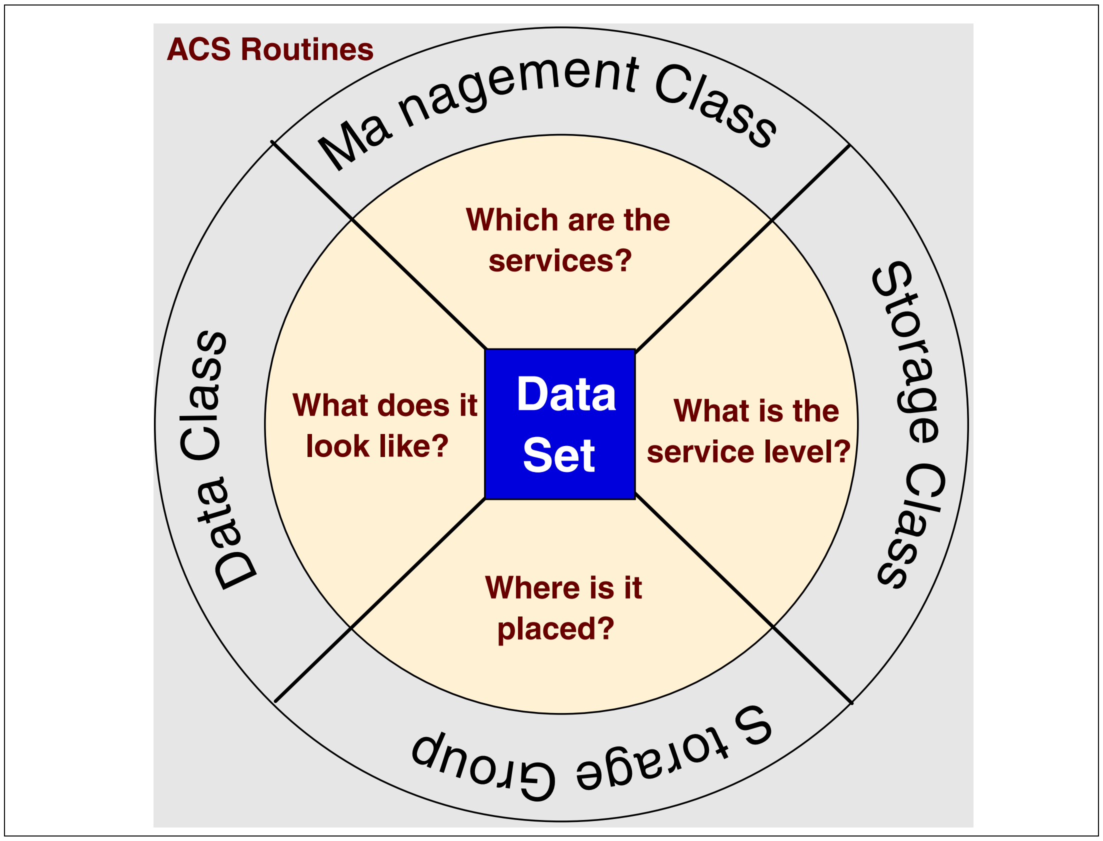


Figure 5-5 Creating SMS policies

Implementing SMS policies

To implement a policy for managing storage, the storage administrator defines classes of space management, performance, and availability requirements for data sets. The storage administrator uses:

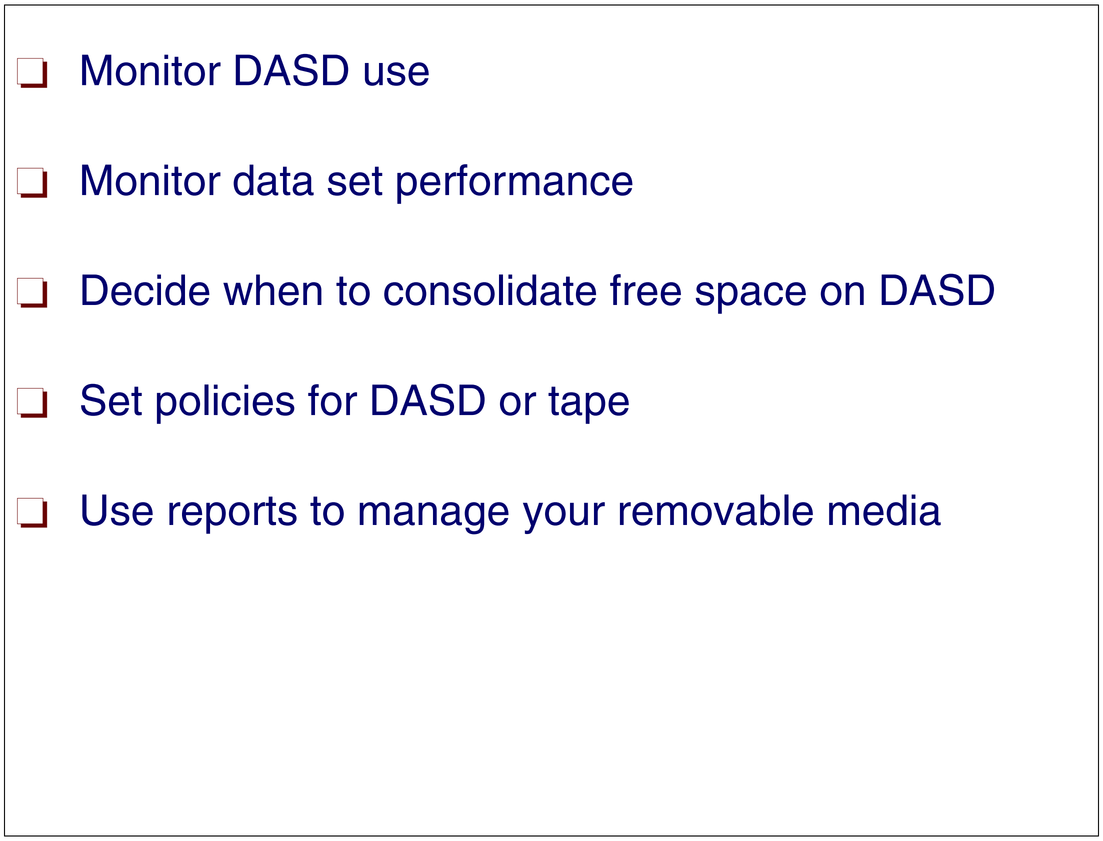
Data class	Data classes are used to define model allocation characteristics for data sets.
Storage class	Storage classes are used to define performance and availability goals.
Management class	Management classes are used to define backup and retention requirements.
Storage group	Storage groups are used to create logical groups of volumes to be managed as a unit.
ACS routines	Automatic Class Selection (ACS) routines are used to assign class and storage group definitions to data sets and objects.

For example, the administrator can define one storage class for data entities requiring high performance, and another for those requiring standard performance. Then, the administrator writes Automatic Class Selection (ACS) routines that use naming conventions or other criteria of your choice to automatically assign the classes that have been defined to data as that data is created. These ACS routines can then be validated and tested.

When the ACS routines are started and the classes (also referred to as constructs) are assigned to the data, SMS uses the policies defined in the classes to apply to the data for the life of the data. Additionally, devices with various characteristics can be pooled together into storage groups, so that new data can be automatically placed on devices that best meet the needs for the data.

DFSMS facilitates all of these tasks by providing menu-driven panels with the *Interactive Storage Management Facility* (ISMF). ISMF panels make it easy to define classes, test and validate ACS routines, and perform other tasks to analyze and manage your storage. Note that many of these functions are available in batch through the NaviQuest tool.

5.6 Monitoring SMS policies



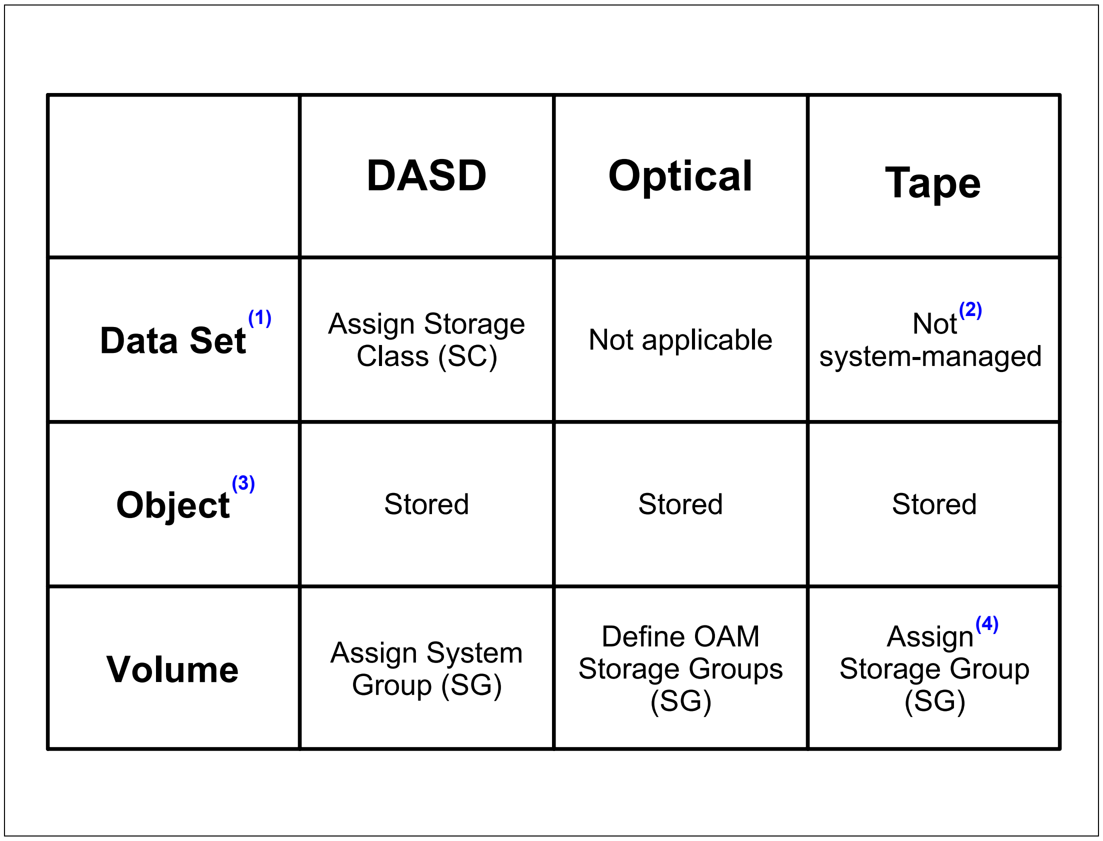
- Monitor DASD use
- Monitor data set performance
- Decide when to consolidate free space on DASD
- Set policies for DASD or tape
- Use reports to manage your removable media

Figure 5-6 Monitoring your SMS policies

Monitoring SMS policies

After storage administrators have established the installation's service levels and implemented policies based on those levels, they can use DFSMS facilities to see if the installation objectives have been met. Information on past use can help to develop more effective storage administration policies and manage growth effectively. The DFSMS Optimizer feature can be used to monitor, analyze, and tune the policies.

5.7 Assigning data to be system-managed



	DASD	Optical	Tape
Data Set[(1)]	Assign Storage Class (SC)	Not applicable	Not[(2)] system-managed
Object[(3)]	Stored	Stored	Stored
Volume	Assign System Group (SG)	Define OAM Storage Groups (SG)	Assign[(4)] Storage Group (SG)

Figure 5-7 How to be system-managed

How to be system-managed

Using SMS, you can automate storage management for individual data sets and objects, and for DASD, optical, and tape volumes. Figure 5-7 shows how a data set, object, DASD volume, tape volume, or optical volume becomes system-managed. The numbers shown in parentheses are associated with the following notes:

1. A DASD data set is *system-managed* if you assign it a storage class. If you do not assign a storage class, the data set is directed to a *non-system-managed* DASD or tape volume - one that is not assigned to a storage group.
2. You can assign a storage class to a tape data set to direct it to a *system-managed* tape volume. However, only the tape volume is considered system-managed, not the data set.
3. Objects are also known as byte-stream data, and this data is used in specialized applications such as image processing, scanned correspondence, and seismic measurements. Object data typically has no internal record or field structure and, once written, the data is not changed or updated. However, the data can be referenced many times during its lifetime. Objects are processed by OAM. Each object has a storage class; therefore, objects are system-managed. The optical or tape volume on which the object resides is also system-managed.
4. Tape volumes are added to tape storage groups in tape libraries when the tape data set is created.

5.8 Using data classes

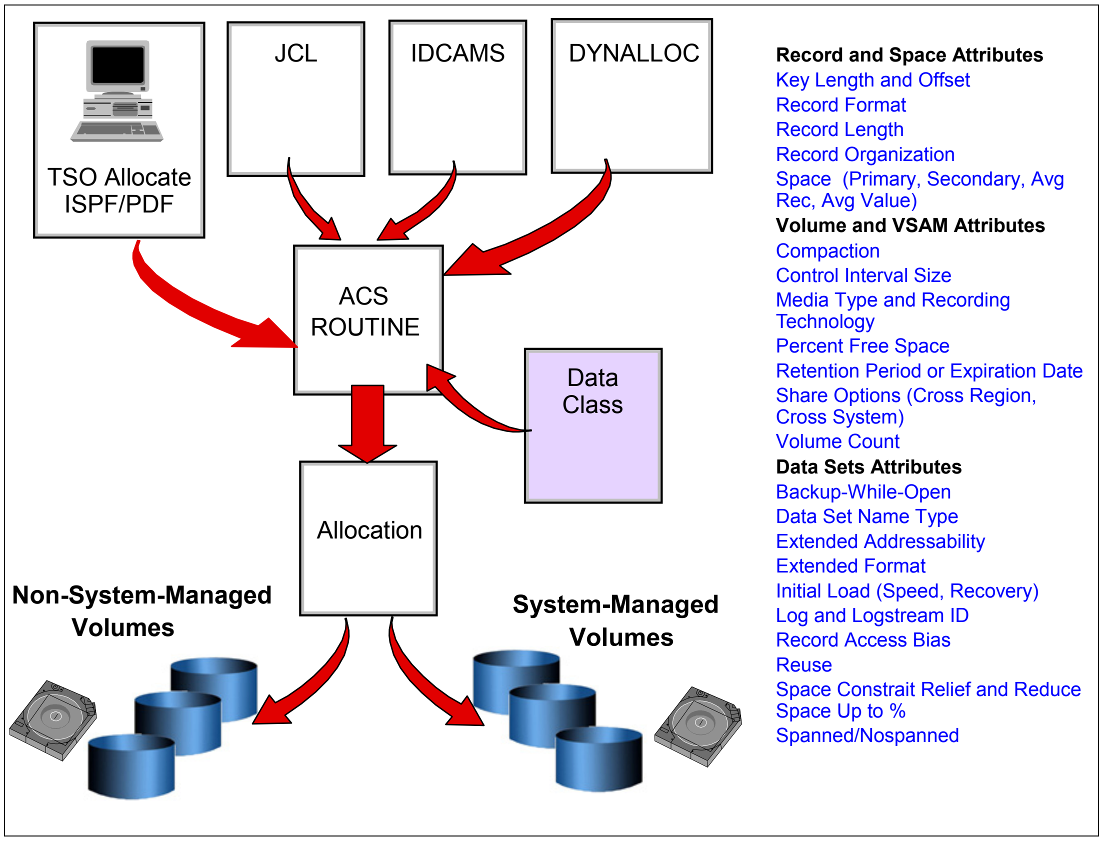


Figure 5-8 Using data classes

Using data classes

A *data class* is a collection of allocation and space attributes that you define. It is used when data sets are created. You can simplify data set allocation for the users by defining data classes that contain standard data set allocation attributes. You can use data classes with *both* system-managed and non-system-managed data sets. However, some data class characteristics, like extended format, are only available for system-managed data sets.

Data class attributes define space and data characteristics that are normally specified on JCL DD statements, TSO/E **ALLOCATE** command, IDCAMS **DEFINE** commands, and dynamic allocation requests. For tape data sets, data class attributes can also specify the type of cartridge and recording method, and if the data is to be compacted. Users then need only specify the appropriate data classes to create standardized data sets.

You can assign a data class through:

- The DATACLAS parameter of a JCL DD statement, **ALLOCATE** or **DEFINE** commands.
- Data class ACS routine to automatically assign a data class when the data set is being created. For example, data sets with the low-level qualifiers LIST, LISTING, OUTLIST, or LINKLIST are usually utility output data sets with similar allocation requirements, and can all be assigned the same data class.

You can override some data set attributes assigned in the data class, but you cannot change the data class name assigned through an ACS routine.

Even though data class is optional, we usually recommend that you assign data classes to system-managed and non-system-managed data. Although the data class is not used after the initial allocation of a data set, the data class name is kept in the catalog entry for system-managed data sets for future reference.

Note: The data class name is not saved for non-system-managed data sets, although the allocation attributes in the data class are used to allocate the data set.

For objects on tape, we recommend that you do not assign a data class via the ACS routines. To assign a data class, specify the name of that data class on the **SETOAM** command.

If you change a data class definition, the changes only affect new allocations. Existing data sets allocated with the data class are not changed.

5.9 Using storage classes

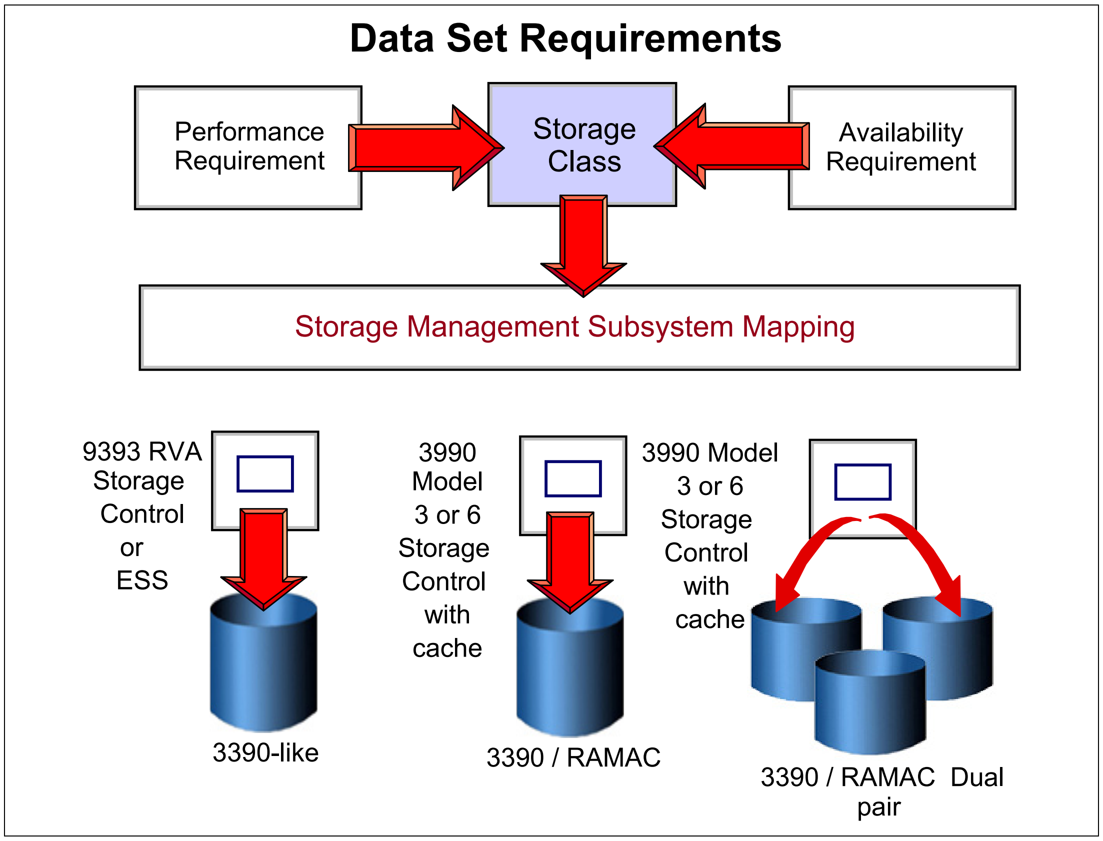

Data Set Requirements
Performance Requirement
Storage Class
Availability Requirement
Storage Management Subsystem Mapping
9393 RVA Storage Control or ESS
3990 Model 3 or 6 Storage Control with cache
3990 Model 3 or 6 Storage Control with cache
3390-like
3390 / RAMAC
3390 / RAMAC Dual pair

Figure 5-9 Choosing volumes that meet availability requirements

Using storage classes

A *storage class* is a collection of performance goals and availability requirements that you define. The storage class is used to select a device to meet those goals and requirements. Only *system-managed* data sets and objects can be assigned to a storage class. Storage classes free users from having to know about the physical characteristics of storage devices and manually placing their data on appropriate devices.

Some of the availability requirements that you specify to storage classes (such as cache and dual copy) can only be met by DASD volumes attached through one of the following storage control units or a similar device:

- 3990-3 or 3990-6
- RAMAC Array Subsystem
- Enterprise Storage Server (ESS)
- DS6000 or DS8000

Figure 5-9 shows storage control unit configurations and their storage class attribute values.

With a storage class, you can assign a data set to dual copy volumes to ensure continuous availability for the data set. With dual copy, two current copies of the data set are kept on separate DASD volumes (by the control unit). If the volume containing the primary copy of the data set is damaged, the companion volume is automatically brought online and the data set continues to be available and current. Remote copy is the same, with the two volumes in distinct control units (generally remote).

You can use the ACCESSIBILITY attribute of the storage class to request that concurrent copy be used when data sets or volumes are backed up.

You can specify an I/O response time objective with storage class by using the millisecond response time (MSR) parameter. During data set allocation, the system attempts to select the closest available volume to the specified performance objective. Also along the data set life, through the use MSR, DFSMS dynamically uses the cache algorithms as DASD Fast Write (DFW) and Inhibit Cache Load (ICL) in order to reach the MSR target I/O response time. This DFSMS function is called *dynamic cache management*.

To assign a storage class to a new data set, you can use:

- The STORCLAS parameter of the JCL DD statement, **`ALLOCATE`** or **`DEFINE`** command
- Storage class ACS routine

For *objects*, the system uses the performance goals you set in the storage class to place the object on DASD, optical, or tape volumes. The storage class is assigned to an object when it is stored or when the object is moved. The ACS routines can override this assignment.

Note: If you change a storage class definition, the changes affect the performance service levels of *existing* data sets that are assigned to that class when the data sets are subsequently opened. However, the definition changes do not affect the location or allocation characteristics of existing data sets.

5.10 Using management classes

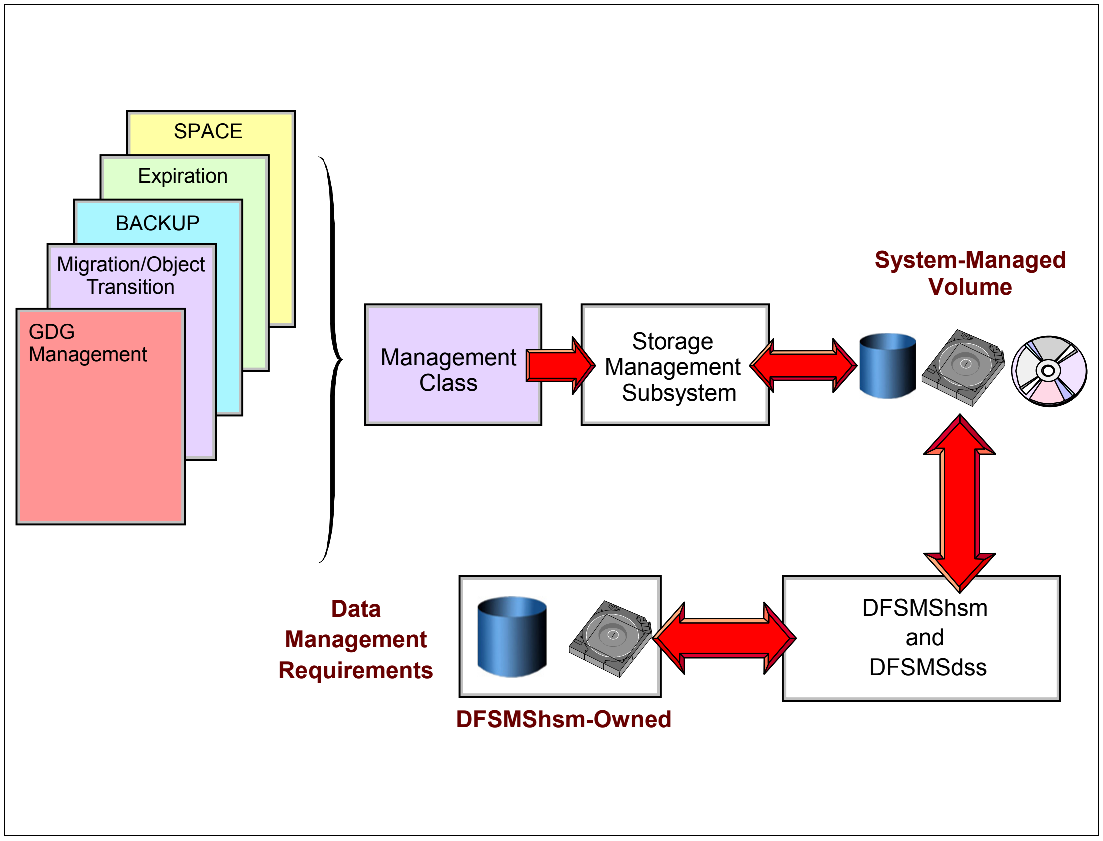


Figure 5-10 Using management classes

Using management classes

A *management class* is a collection of management attributes that you define. The attributes defined in a management class are related to:

- Expiration date
- Migration criteria
- GDG management
- Backup of data set
- Object Class Transition Criteria
- Aggregate backup

Management classes let you define management requirements for individual data sets, rather than defining the requirements for entire volumes. All the data set functions described in the management class are executed by DFSMShsm and DFSMSdss programs. Figure 5-11 on page 237 shows the sort of functions an installation can define in a management class.

To assign a management class to a new data set, you can use:

- The MGMTCLAS parameter of the JCL DD statement, **`ALLOCATE`** or **`DEFINE`** command
- The management class ACS routine to automatically assign management classes to new data sets

The ACS routine can override the management class specified in JCL, **`ALLOCATE`** or **`DEFINE`** command.You cannot override management class *attributes* via JCL or command parameters.

If you do not explicitly assign a management class to a system-managed data set or object, the system uses the *default management class*. You can define your own default management class when you define your SMS base configuration.

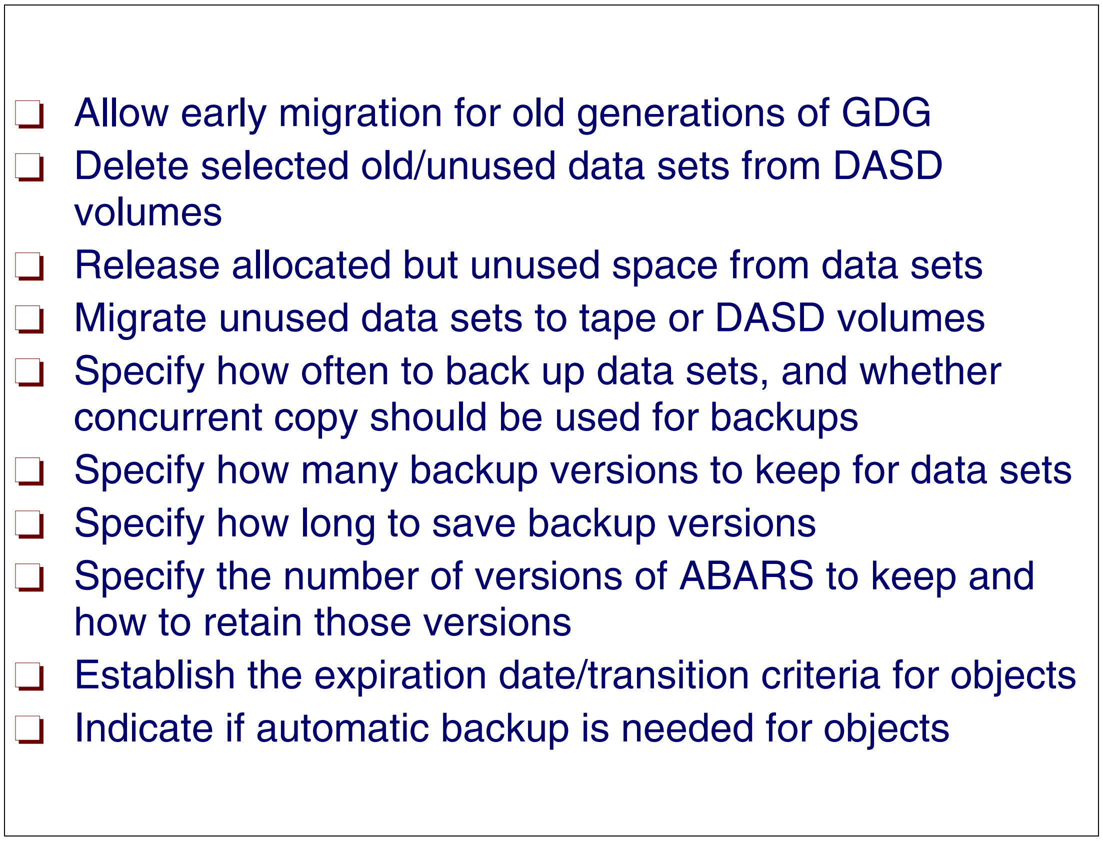
Note: If you change a management class definition, the changes affect the management requirements of *existing data sets* and *objects* that are assigned that class. You can reassign management classes when data sets are renamed.

For objects, you can:

- Assign a management class when it is stored, or
- Assign a new management class when the object is moved, or
- Change the management class by using the OAM Application Programming Interface (OSREQ CHANGE function)

The ACS routines can override this assignment for objects.

5.11 Management class functions



- ❑ Allow early migration for old generations of GDG
- ❑ Delete selected old/unused data sets from DASD volumes
- ❑ Release allocated but unused space from data sets
- ❑ Migrate unused data sets to tape or DASD volumes
- ❑ Specify how often to back up data sets, and whether concurrent copy should be used for backups
- ❑ Specify how many backup versions to keep for data sets
- ❑ Specify how long to save backup versions
- ❑ Specify the number of versions of ABARS to keep and how to retain those versions
- ❑ Establish the expiration date/transition criteria for objects
- ❑ Indicate if automatic backup is needed for objects

Figure 5-11 Management class functions

Management class functions

By classifying data according to management requirements, an installation can define unique management classes to fully automate data set and object management. For example:

- Control the migration of CICS user databases, DB2 user databases and archive logs.
- Test systems and their associated data sets.
- IMS™ archive logs.
- Specify that DB2 image copies, IMS image copies and change accumulation logs be written to primary volumes and then migrated directly to migration level 2 tape volumes.
- For objects, define when an object is eligible for a change in its performance objectives or management characteristics. For example, after a certain number of days an installation might want to move an object from a high performance DASD volume to a slower optical volume.

  Management class can also be used to specify that the object should have a backup copy made when the OAM Storage Management Component (OSMC) is executing.

When changing a management class definition, the changes affect the management requirements of existing data sets and objects that are assigned to that class.

5.12 Using storage groups

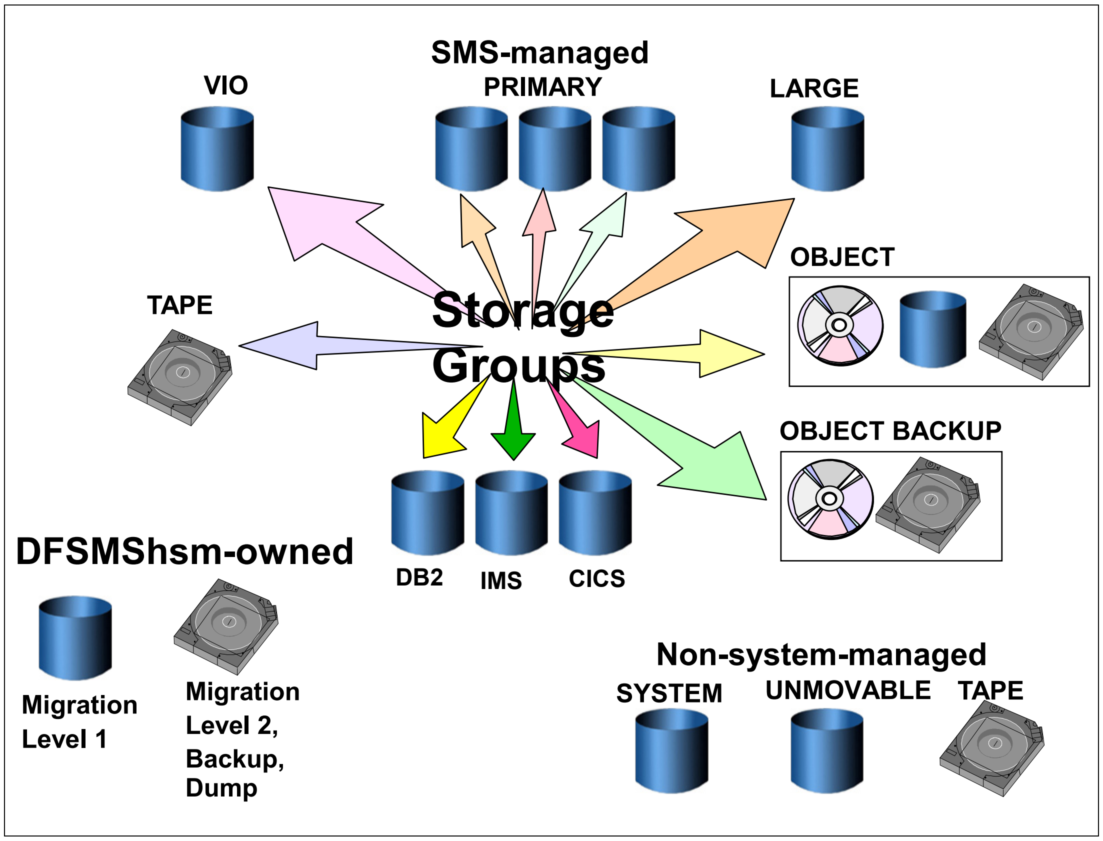


Figure 5-12 Grouping storage volumes for specific purposes

Storage groups

A *storage group* is a collection of storage volumes and attributes that you define. The collection can be a group of:

- System paging volumes
- DASD volumes
- Tape volumes
- Optical volumes
- Combination of DASD and optical volumes that look alike
- DASD, tape, and optical volumes treated as a single object storage hierarchy

Storage groups, along with storage classes, help reduce the requirement for users to understand the physical characteristics of the storage devices which contain their data.

In a tape environment, you can also use tape storage groups to direct a new tape data set to an automated or manual tape library.

DFSMShsm uses some of the storage group attributes to determine if the volumes in the storage group are eligible for automatic space or availability management.

Figure 5-12 shows an example of how an installation can group storage volumes according to their objective. In this example:

- SMS-managed DASD volumes are grouped into storage groups so that primary data sets, large data sets, DB2 data, IMS data, and CICS data are all separated.

- The VIO storage group uses system paging volumes for small temporary data sets.
- The TAPE storage groups are used to group tape volumes that are held in tape libraries.
- The OBJECT storage group can span optical, DASD, and tape volumes.
- The OBJECT BACKUP storage group can contain either optical or tape volumes within one OAM invocation.
- Some volumes are not system-managed.
- Other volumes are owned by DFSMShsm for use in data backup and migration. DFSMShsm migration level 2 tape cartridges can be system-managed if you assign them to a tape storage group.

Note: A storage group is assigned to a data set *only* through the storage group ACS routine. Users cannot specify a storage group when they allocate a data set, although they *can* specify a unit and volume.

Whether or not to honor a user's unit and volume request is an installation decision, but we recommend that you discourage users from directly requesting specific devices. It is more effective for users to specify the logical storage requirements of their data by storage and management class, which the installation can then verify in the ACS routines.

For objects, there are two types of storage groups, OBJECT and OBJECT BACKUP. An OBJECT storage group is assigned by OAM when the object is stored; the storage group ACS routine can override this assignment. There is only one OBJECT BACKUP storage group, and all backup copies of all objects are assigned to this storage group.

SMS volume selection

SMS determines which volumes are used for data set allocation by developing a list of all volumes from the storage groups assigned by the storage group ACS routine. Volumes are then either removed from further consideration or flagged as the following:

Primary	Volumes online, below threshold, that meet all the specified criteria in the storage class.
Secondary	Volumes that do not meet all the criteria for primary volumes.
Tertiary	When the number of volumes in the storage group is less than the number of volumes that are requested.
Rejected	Volumes that do not meet the required specifications. They are not candidates for selection.

SMS starts volume selection from the primary list; if no volumes are available, SMS selects from the secondary; and, if no secondary volumes are available, SMS selects from the tertiary list.

SMS interfaces with the system resource manager (SRM) to select from the eligible volumes in the primary list. SRM uses device delays as one of the criteria for selection, and does not prefer a volume if it is already allocated in the jobstep. This is useful for batch processing when the data set is accessed immediately after creation.

SMS does not use SRM to select volumes from the secondary or tertiary volume lists. It uses a form of randomization to prevent skewed allocations in instances such as when new volumes are added to a storage group, or when the free space statistics are not current on volumes.

For a striped data set, when multiple storage groups are assigned to an allocation, SMS examines each storage group and selects the one that offers the largest number of volumes attached to unique control units. This is called *control unit separation.* Once a storage group has been selected, SMS selects the volumes based on available space, control unit separation, and performance characteristics if they are specified in the assigned storage class.

5.13 Using aggregate backup and recovery support (ABARS)

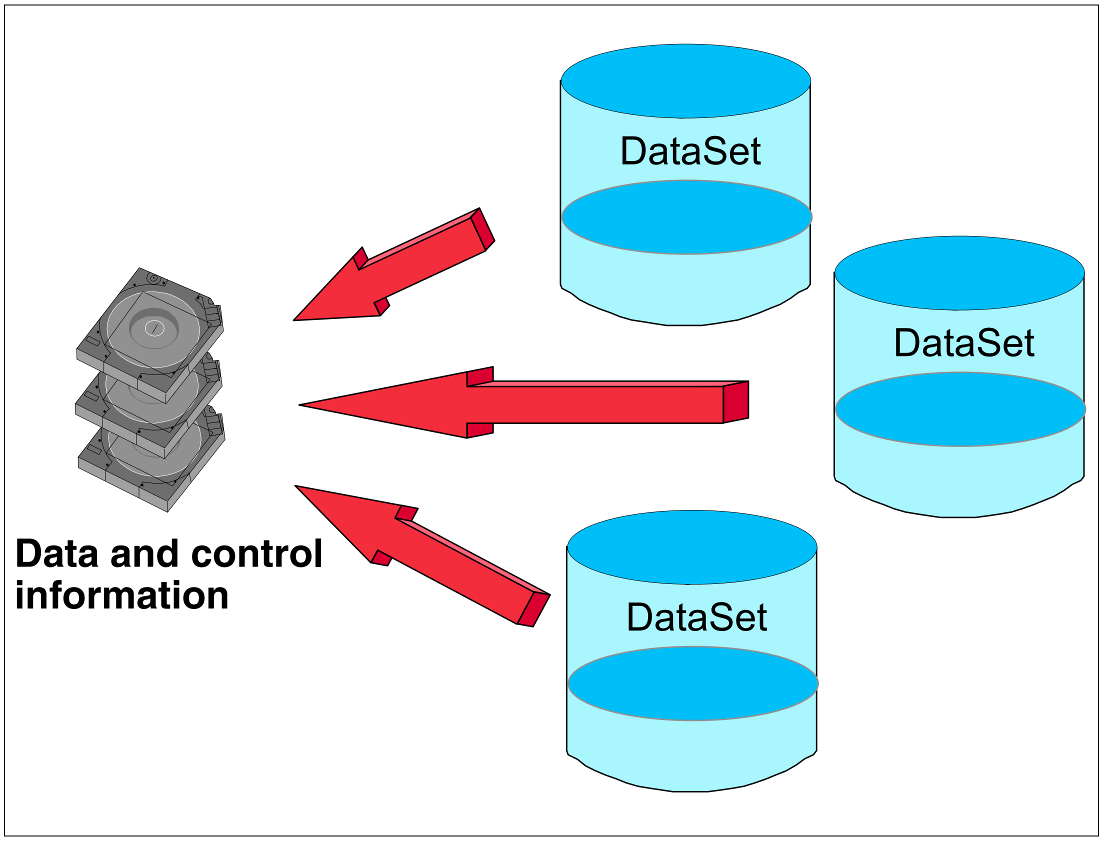


Figure 5-13 ABARS

Aggregate backup and recovery support (ABARS)

Aggregate backup and recovery support, also called *application backup application recovery support*, is a command-driven process to back up and recover any user-defined group of data sets that are vital to your business. An *aggregate group* is a collection of related data sets and control information that has been pooled to meet a defined backup or recovery strategy. If a disaster occurs, you can use these backups at a remote or local site to recover critical applications.

The user-defined group of data sets can be those belonging to an application, or any combination of data sets that you want treated as a separate entity. Aggregate processing enables you to:

- Back up and recover data sets by application, to enable business to resume at a remote site if necessary
- Move applications in a non-emergency situation in conjunction with personnel moves or workload balancing
- Duplicate a problem at another site

You can use aggregate groups as a supplement to using management class for applications that are critical to your business. You can associate an aggregate group with a management class. The management class specifies backup attributes for the aggregate group, such as the copy technique for backing up DASD data sets on primary volumes, the number of aggregate versions to retain, and how long to retain versions. Aggregate groups simplify the control of backup and recovery of critical data sets and applications.

Although SMS must be used on the system where the backups are performed, you can recover aggregate groups to systems that are not using SMS, provided that the groups do not contain data that requires that SMS be active, such as PDSEs. You can use aggregate groups to transfer applications to other data processing installations, or to migrate applications to newly-installed DASD volumes. You can transfer the application's migrated data, along with its active data, without recalling the migrated data.

5.14 Automatic Class Selection (ACS) routines

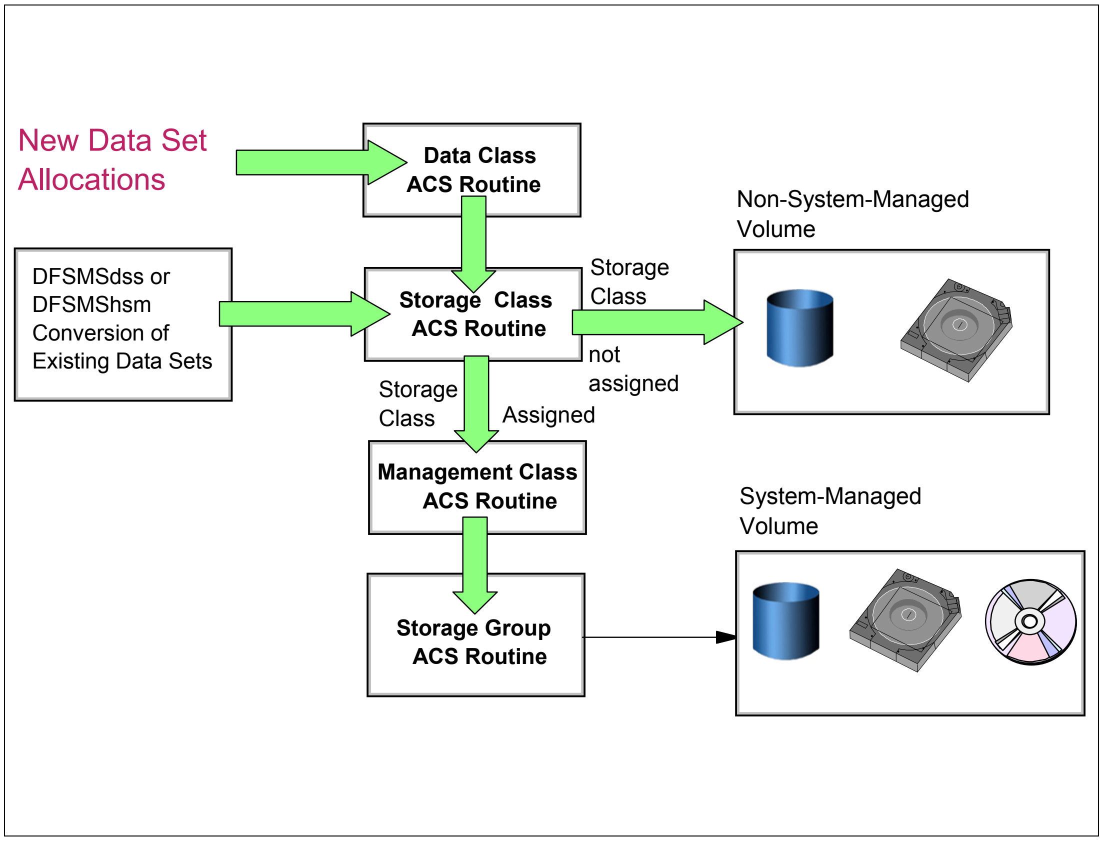

New Data Set Allocations
Data Class ACS Routine
DFSMSdss or DFSMShsm Conversion of Existing Data Sets
Storage Class ACS Routine
Storage Class not assigned
Non-System-Managed Volume
Storage Class Assigned
Management Class ACS Routine
Storage Group ACS Routine
System-Managed Volume

Figure 5-14 Using ACS routines

Using Automatic Class Selection routines

You use automatic class selection (ACS) routines to assign classes (data, storage, and management) and storage group definitions to data sets, database data, and objects. You write ACS routines using the ACS language, which is a high-level programming language. Once written, you use the ACS translator to translate the routines to object form so they can be stored in the SMS configuration.

The ACS language contains a number of read-only variables, which you can use to analyze new data allocations. For example, you can use the read-only variable &DSN to make class and group assignments based on data set or object collection name, or &LLQ to make assignments based on the low-level qualifier of the data set or object collection name.

With z/OS V1R6, you can use a new ACS routine read-only security label variable, &SECLABL, as input to the ACS routine. A security label is a name used to represent an association between a particular security level and a set of security categories. It indicates the minimum level of security required to access a data set protected by this profile.

Note: You cannot alter the value of read-only variables.

You use the four read-write variables to assign the class or storage group you determine for the data set or object, based on the routine you are writing. For example, you use the &STORCLAS variable to assign a storage class to a data set or object.

For a detailed description of the ACS language and its variables, see *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

For each SMS configuration, you can write as many as four routines: one each for data class, storage class, management class, and storage group. Use ISMF to create, translate, validate, and test the routines.

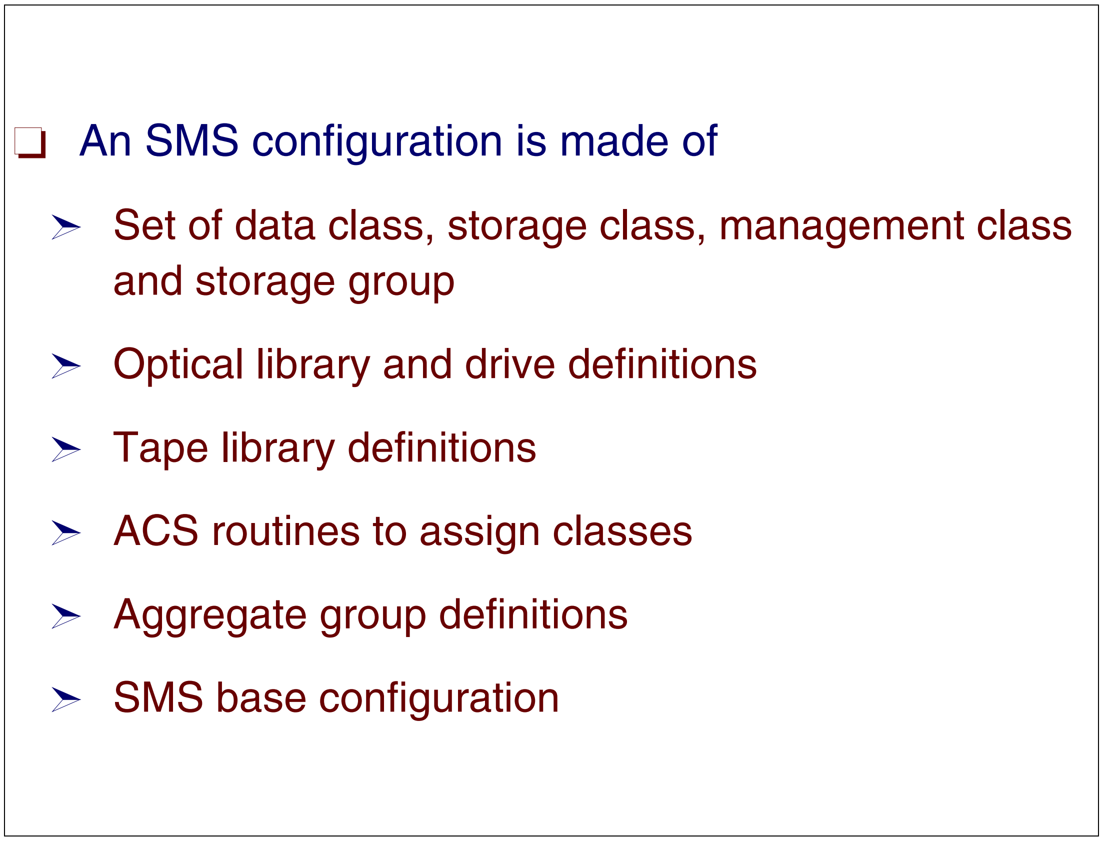
Processing order of ACS routines

Figure 5-14 on page 242 shows the order in which ACS routines are processed. Data can become system-managed if the storage class routine assigns a storage class to the data, or if it allows a user-specified storage class to be assigned to the data. If this routine does not assign a storage class to the data, the data cannot reside on a system-managed volume.

Because data allocations, whether dynamic or through JCL, are processed through ACS routines, you can enforce installation standards for data allocation on system-managed and non-system-managed volumes. ACS routines also enable you to override user specifications for data, storage, and management class, and requests for specific storage volumes.

You can use the ACS routines to determine the SMS classes for data sets created by the Distributed FileManager/MVS. If a remote user does not specify a storage class, and if the ACS routines decide that the data set should not be system-managed, the Distributed FileManager/MVS terminates the creation process immediately and returns an error reply message to the source. Therefore, when you construct your ACS routines, consider the potential data set creation requests of remote users.

5.15 SMS configuration



- An SMS configuration is made of
  - Set of data class, storage class, management class and storage group
  - Optical library and drive definitions
  - Tape library definitions
  - ACS routines to assign classes
  - Aggregate group definitions
  - SMS base configuration

Figure 5-15 Defining the SMS configuration

SMS configuration

An SMS configuration is composed of:

- A set of data class, management class, storage class, and storage group
- ACS routines to assign the classes and groups
- Optical library and drive definitions
- Tape library definitions
- Aggregate group definitions
- SMS base configuration, that contains information such as:
  - Default management class
  - Default device geometry
  - The systems in the installation for which the subsystem manages storage

The SMS configuration is stored in SMS control data sets, which are VSAM linear data sets. You must define the control data sets before activating SMS. SMS uses the following types of control data sets:

- Source Control Data Set (SCDS)
- Active Control Data Set (ACDS)
- Communications Data Set (COMMDS)

SMS complex (SMSplex)

A collection of systems or system groups that share a common configuration is called an SMS complex. All systems in an SMS complex share an ACDS and a COMMDS. The systems or system groups that share the configuration are defined to SMS in the SMS base configuration. The ACDS and COMMDS must reside on a shared volume, accessible for all systems in the SMS complex.

5.16 SMS control data sets

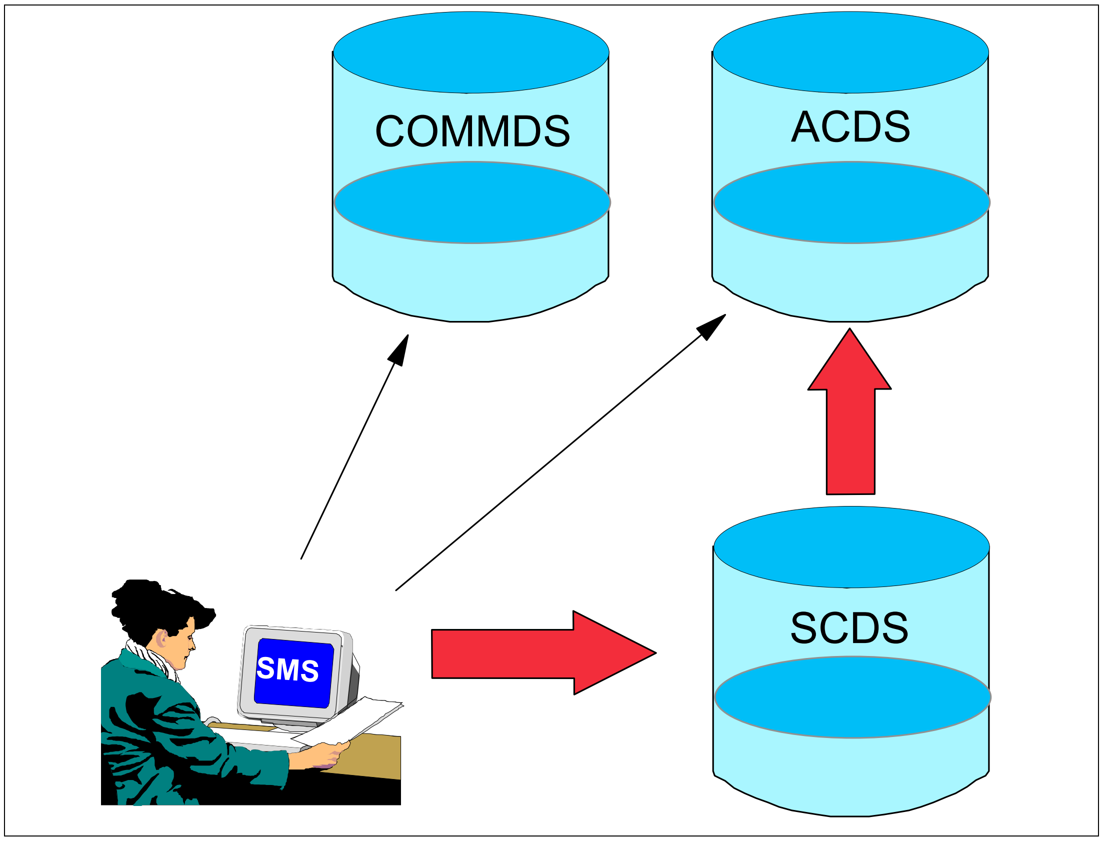


Figure 5-16 SMS control data sets

SMS control data sets

SMS stores its class and group definitions, translated ACS routines, and system information in three control data sets.

Source Control Data Set (SCDS)

The SCDS contains SMS classes, groups, and translated ACS routines that define a *single storage management policy*, called an SMS configuration. You can have several SCDSs, but only one can be used to activate the SMS configuration.

You use the SCDS to develop and test but, before activating a configuration, retain at least one prior configuration should you need to regress to it because of error. The SCDS is never used to manage allocations.

Active Control Data Set (ACDS)

The ACDS is the system's active copy of the current SCDS. When you activate a configuration, SMS copies the existing configuration from the specified SCDS into the ACDS. By using copies of the SMS classes, groups, volumes, optical libraries, optical drives, tape libraries, and ACS routines rather than the originals, you can change the current storage management policy without disrupting it. For example, while SMS uses the ACDS, you can:

- Create a copy of the ACDS
- Create a backup copy of an SCDS

- Modify an SCDS
- Define a new SCDS

We recommend that you have extra ACDSs in case a hardware failure causes the loss of your primary ACDS. It must reside on a shared device, accessible to all systems, to ensure that they share a common view of the active configuration. Do not have the ACDS reside on the same device as the COMMDS or SCDS. Both the ACDS and COMMDS are needed for SMS operation across the complex. Separation protects against hardware failure. You should also create a backup ACDS in case of hardware failure or accidental data loss or corruption.

Communications Data Set (COMMDS)

The COMMDS data set contains the name of the ACDS and storage group volume statistics. It enables communication between SMS systems in a multisystem environment. The COMMDS also contains space statistics, SMS status, and MVS status for each system-managed volume.

The COMMDS must reside on a shared device accessible to all systems. However, do not allocate it on the same device as the ACDS. Create a spare COMMDS in case of a hardware failure or accidental data loss or corruption. SMS activation fails if the COMMDS is unavailable.

5.17 Implementing DFSMS

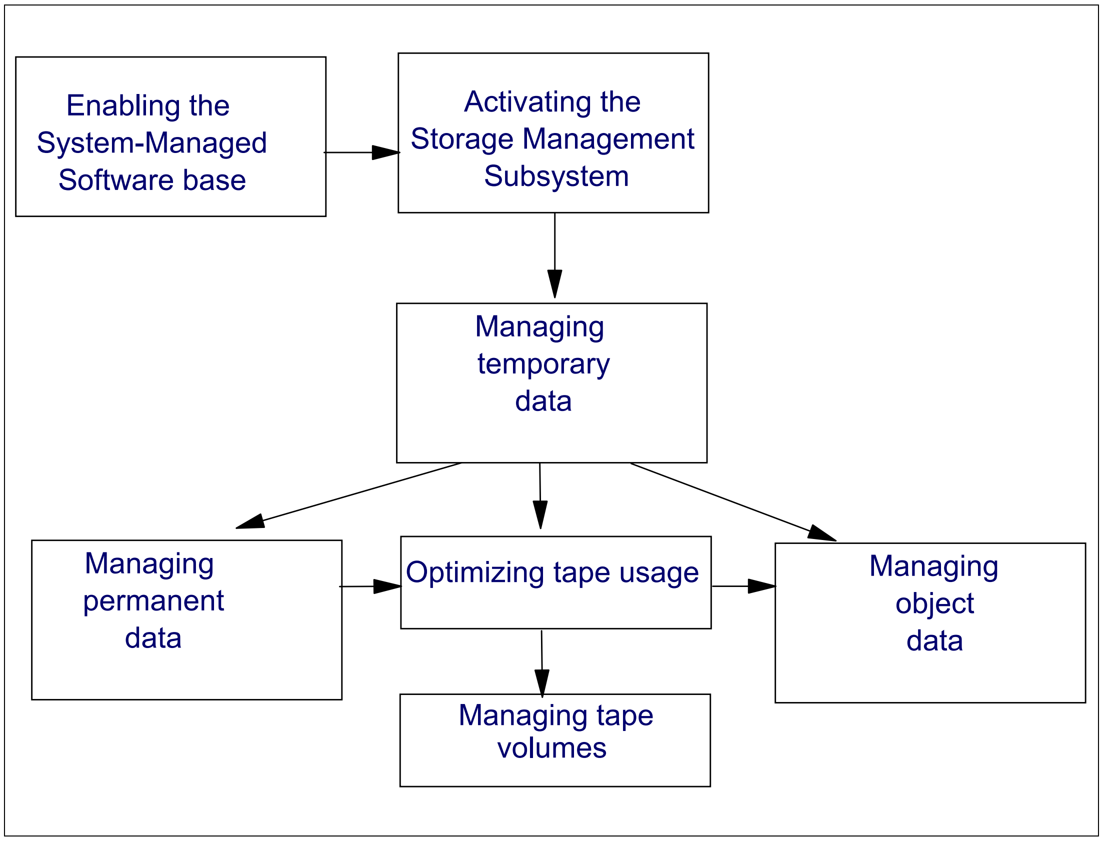


Figure 5-17 SMS implementation phases

Implementing DFSMS

You can implement SMS to fit your specific needs. You do not have to implement and use all of the SMS functions. Rather, you can implement the functions you are most interested in first. For example, you can:

- Set up a storage group to only exploit the functions provided by extended format data sets, such as striping, system-managed buffering (SMB), partial release, and so on.
- Put some of your data in a pool of one or more storage groups and assign them policies at the storage group level to implement DFSMShsm operations in stages.
- Exploit VSAM record level sharing (RLS).

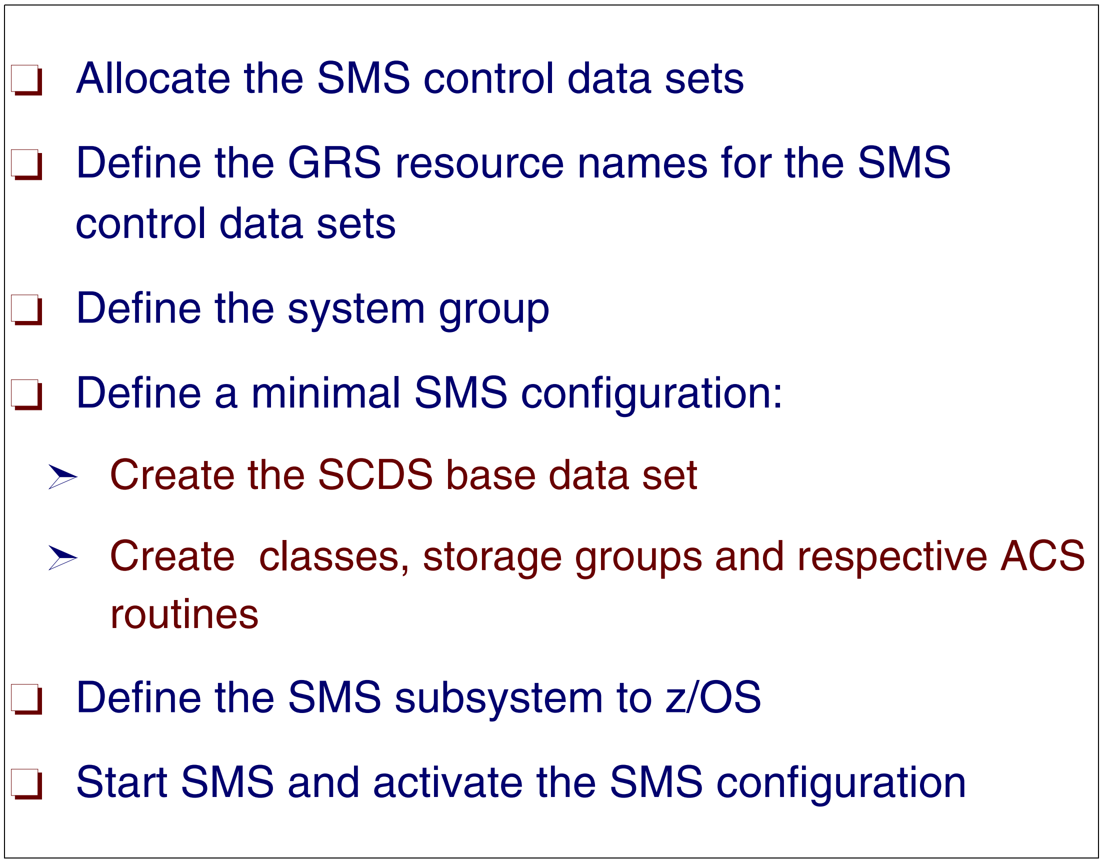
DFSMS implementation phases

There are five major DFSMS implementation phases:

- Enabling the software base
- Activating the storage management subsystem
- Managing temporary data
- Managing permanent data
- Managing tape data

In this book, we present an overview of the steps needed to activate, and manage data with, a minimal SMS configuration, without affecting your JCL or data set allocations. To implement DFSMS in your installation, however, refer to *z/OS DFSMS Implementing System-Managed Storage,* SC26-7407.

5.18 Steps to activate a minimal SMS configuration



- ❑ Allocate the SMS control data sets
- ❑ Define the GRS resource names for the SMS control data sets
- ❑ Define the system group
- ❑ Define a minimal SMS configuration:
  - ➢ Create the SCDS base data set
  - ➢ Create classes, storage groups and respective ACS routines
- ❑ Define the SMS subsystem to z/OS
- ❑ Start SMS and activate the SMS configuration

Figure 5-18 Steps to activate a minimal SMS configuration

Steps to activate a minimal SMS configuration

Activating a minimal configuration lets you experience managing an SMS configuration without affecting your JCL or data set allocations. This establishes an operating environment for the storage management subsystem, without data sets becoming system-managed.

The minimal SMS configuration consists of the following elements:

- A base configuration
- A storage class definition
- A storage group containing at least one volume
- A storage class ACS routine
- A storage group ACS routine

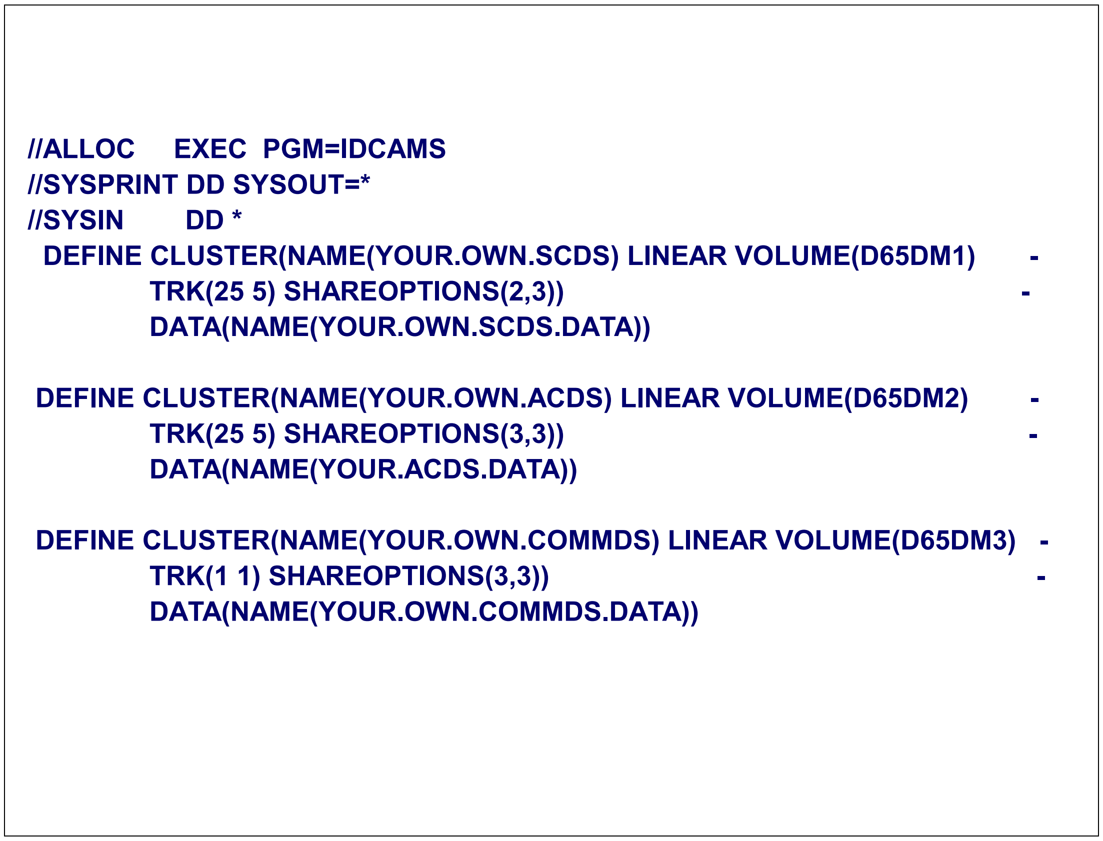
All of these elements are required for a valid SMS configuration, except for the storage class ACS routine.

The steps needed to activate the minimal configuration are presented in Figure 5-18. When implementing DFSMS, beginning by implementing a minimal configuration allows you to:

- Gain experience with ISMF applications for the storage administrator, since you use ISMF applications to define and activate the SMS configuration.

- Gain experience with the operator commands that control operation of resources controlled by SMS.
- Learn how the SMS base configuration can affect allocations for non-system-managed data sets. The base configuration contains installation defaults for data sets:
- For non-system-managed data sets, you can specify default device geometry to ease the conversion from device-dependent space calculations to the device-independent method implemented by SMS.
- For system-managed data sets, you can specify a default management class to be used for data sets that are not assigned a management class by your management class ACS routine.
- Use simplified JCL.
- Implement allocation standards, since you can develop a data class ACS routine to enforce your standards.

5.19 Allocating SMS control data sets



```
//ALLOC     EXEC  PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN       DD *
  DEFINE CLUSTER(NAME(YOUR.OWN.SCDS) LINEAR VOLUME(D65DM1)      -
              TRK(25 5) SHAREOPTIONS(2,3))                       -
              DATA(NAME(YOUR.OWN.SCDS.DATA))

 DEFINE CLUSTER(NAME(YOUR.OWN.ACDS) LINEAR VOLUME(D65DM2)       -
              TRK(25 5) SHAREOPTIONS(3,3))                        -
              DATA(NAME(YOUR.ACDS.DATA))

 DEFINE CLUSTER(NAME(YOUR.OWN.COMMDS) LINEAR VOLUME(D65DM3)   -
              TRK(1 1) SHAREOPTIONS(3,3))                         -
              DATA(NAME(YOUR.OWN.COMMDS.DATA))
```

Figure 5-19 Using IDCAMS to create SMS control data sets

Calculating the SCDS and ACDS sizes

The size of the ACDS and SCDS may allow constructs for up to 32 systems. Be sure to allocate sufficient space for the ACDS and SCDS, since insufficient ACDS size can cause errors such as failing SMS activation. See *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402 for the formula used to calculate the appropriate SMS control data set size.

Calculating the COMMDS size

The size of the communications data set (COMMDS) increased in DFSMS 1.3, because the amount of space required to store system-related information for each volume increased. To perform a precise calculation of the COMMDS size, use the formula provided in *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

Defining the control data sets

After you have calculated their respective sizes, define the SMS control data sets using access method services. The SMS control data sets are VSAM linear data sets and you define them using the IDCAMS `DEFINE` command, as shown in Figure 5-19. Because these data sets are allocated *before* SMS is activated, space is allocated in tracks. Allocations in KBs or MBs are only supported when SMS is active.

Specify SHAREOPTIONS(2,3) only for the SCDS. This lets one update-mode user operate simultaneously with other read-mode users between regions.

Specify SHAREOPTIONS(3,3) for the ACDS and COMMDS. These data sets must be shared between systems that are managing a shared DASD configuration in a DFSMS environment.

Define GRS resource names for active SMS control data sets

If you plan to share SMS control data sets between systems, consider the effects of multiple systems sharing these data sets. Access is serialized by the use of RESERVE, which locks out access to the *entire device volume* from other systems until the RELEASE is issued by the task using the resource. This is undesirable, especially when there are other data sets on the volume.

A RESERVE is issued when SMS is updating:

- The COMMDS with space statistics at the expiration time interval specified in the IGDSMSxx PARMLIB member
- The ACDS due to changes in the SMS configuration

Place the resource name IGDCDSXS in the *RESERVE conversion RNL* as a generic entry to convert the RESERVE/RELEASE to an ENQueue/DEQueue. This minimizes delays due to contention for resources and prevents deadlocks associated with the **VARY SMS** command.

Important: If there are multiple SMS complexes within a global resource serialization complex, be sure to use unique COMMDS and ACDS data set names to prevent false contention.

For information about allocating COMMDS and ACDS data set names, see *z/OS DFSMS Implementing System-Managed Storage,* SC26-7407.

5.20 Defining the SMS base configuration

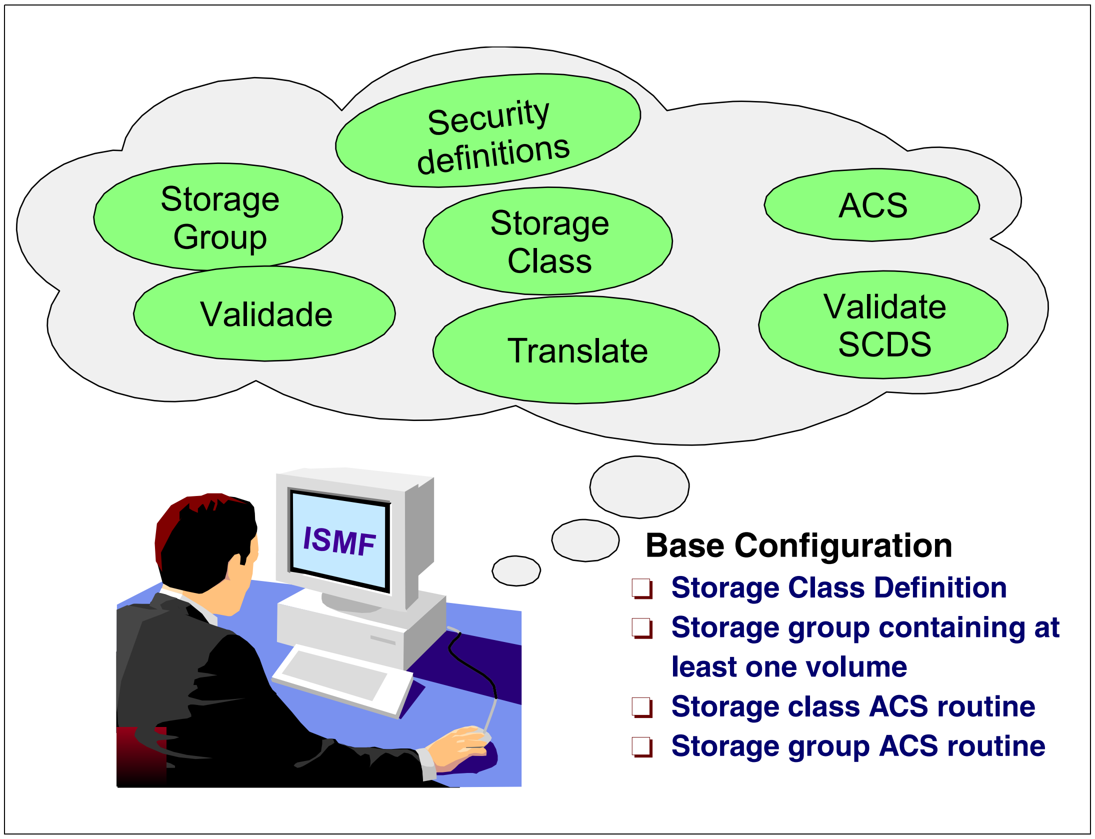

Security definitions
Storage Group
Storage Class
ACS
Validade
Translate
Validate SCDS
ISMF
Base Configuration
❑ Storage Class Definition
❑ Storage group containing at least one volume
❑ Storage class ACS routine
❑ Storage group ACS routine

Figure 5-20 Minimal SMS configuration

Protecting the DFSMS environment

Before defining the SMS base configuration, you have to protect access to the SMS control data sets, programs, and functions. For example, some functions in ISMF are related only to storage administration tasks and you must protect your storage environment from unauthorized access. You can protect the DFSMS environment with RACF.

RACF controls access to the following resources:

- System-managed data sets
- SMS control data sets
- SMS functions and commands
- Fields in the RACF profile
- SMS classes
- ISMF functions

For more information, refer to *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

Defining the system group

A *system group* is a group of systems within an SMS complex that have similar connectivity to storage groups, libraries, and volumes. When a Parallel Sysplex name is specified and used as a system group name, the name applies to all systems in the Parallel Sysplex except for those systems defined as part of the Parallel Sysplex that are explicitly named in the SMS base configuration. The system group is defined using ISMF when defining the base configuration.

Defining the SMS base configuration

After creating the SCDS data set with IDCAMS and setting up the security to the DFSMS environment, you use the ISMF **Control Data Set** option to define the SMS base configuration, which contains information such as:

- Default management class
- Default device geometry
- The systems in the installation for which SMS manages storage using that configuration

To define a minimal configuration, you must do the following:

- Define a storage class.
- Define a storage group containing at least one volume. (The volume does not have to exist, as long as you do not direct allocations to either the storage group or the volume.)
- Create their respective ACS routines.

Defining a data class, a management class, and creating their respective ACS routines are not required for a valid SCDS. However, because of the importance of the *default management class*, we recommend that you include it in your minimal configuration.

For a detailed description of SMS classes and groups, see *z/OS DFSMS Implementing System-Managed Storage,* SC26-7407.

The DFSMS product tape contains a set of sample ACS routines. The appendix of *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402 contains sample definitions of the SMS classes and groups that are used in the sample ACS routines. The starter set configuration can be used as a model for your own SCDS. For a detailed description of base configuration attributes and how to use ISMF to define its contents, see *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

Defining the storage class

You must define at least one storage class name to SMS. Because a minimal configuration does not include any system-managed volumes, no performance or availability information need be contained in the minimal configuration's storage class. Specify an artificial storage class, NONSMS. This class is later used by the storage administrator to create non-system-managed data sets on an exception basis.

In the storage class ACS routine, the &STORCLAS variable is set to a null value to prevent users from coding a storage class in JCL before you want to have system-managed data sets.

You define the class using ISMF. Select **Storage Class** in the primary menu. Then you can define the class, NONSMS, in your configuration in one of two ways:

- Select option **3 Define** in the Storage Class Application Selection panel. The CDS Name field must point to the SCDS you are building.
- Select option **1 Display** in the Storage Class Application Selection panel. The CDS Name field must point to the starter set SCDS. Then, in the displayed panel, use the `COPY` line operator to copy the definition of NONSMS from the starter set SCDS to your own SCDS.

Defining the storage group

You must define at least one pool storage group name to SMS, and at least one volume serial number to this storage group. A storage group with no volumes defined is not valid. This volume serial number should be for a *nonexistent volume* to prevent the occurrence of JCL errors from jobs accessing data sets using a specific volume serial number.

Defining a non-existent volume lets you activate SMS without having any system-managed volumes. No data sets are system-managed at this time. This condition provides an opportunity to experiment with SMS without any risk to your data.

Define a storage group (for example, NOVOLS) in your SCDS. A name like NOVOLS is useful because you know it does not contain valid volumes.

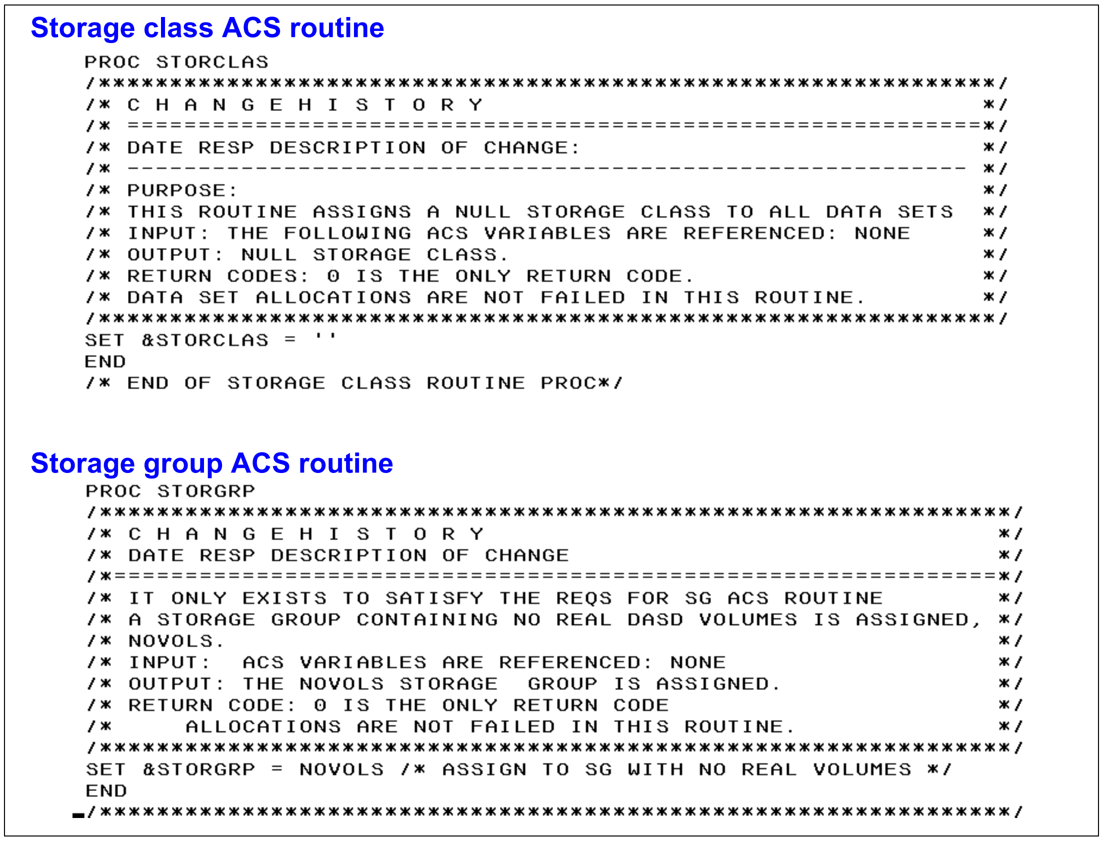
Defining the default management class

Define a default management class and name it STANDEF to correspond with the entry in the base configuration. We recommend that you specifically assign all system-managed data to a management class. If you do not supply a default, DFSMShsm uses two days on primary storage, and 60 days on migration level 1 storage, as the default.

No management classes are assigned when the minimal configuration is active. Definition of this default is done here to prepare for the managing permanent data implementation phase.

The management class, STANDEF, is defined in the starter set SCDS. You can copy its definition to your own SCDS in the same way as the storage class, NONSMS.

5.21 Creating ACS routines



Storage class ACS routine

```
PROC STORCLAS
/****************************************************************/
/* C H A N G E H I S T O R Y                                    */
/* ============================================================*/
/* DATE RESP DESCRIPTION OF CHANGE:                             */
/* ---------------------------------------------------------- */
/* PURPOSE:                                                     */
/* THIS ROUTINE ASSIGNS A NULL STORAGE CLASS TO ALL DATA SETS  */
/* INPUT: THE FOLLOWING ACS VARIABLES ARE REFERENCED: NONE      */
/* OUTPUT: NULL STORAGE CLASS.                                  */
/* RETURN CODES: 0 IS THE ONLY RETURN CODE.                     */
/* DATA SET ALLOCATIONS ARE NOT FAILED IN THIS ROUTINE.         */
/****************************************************************/
SET &STORCLAS = ''
END
/* END OF STORAGE CLASS ROUTINE PROC*/
```

Storage group ACS routine

```
PROC STORGRP
/****************************************************************/
/* C H A N G E H I S T O R Y                                    */
/* DATE RESP DESCRIPTION OF CHANGE                              */
/*============================================================*/
/* IT ONLY EXISTS TO SATISFY THE REQS FOR SG ACS ROUTINE        */
/* A STORAGE GROUP CONTAINING NO REAL DASD VOLUMES IS ASSIGNED, */
/* NOVOLS.                                                      */
/* INPUT:  ACS VARIABLES ARE REFERENCED: NONE                   */
/* OUTPUT: THE NOVOLS STORAGE  GROUP IS ASSIGNED.               */
/* RETURN CODE: 0 IS THE ONLY RETURN CODE                       */
/*     ALLOCATIONS ARE NOT FAILED IN THIS ROUTINE.              */
/****************************************************************/
SET &STORGRP = NOVOLS /* ASSIGN TO SG WITH NO REAL VOLUMES */
END
/****************************************************************/
```

Figure 5-21 Sample ACS routines for a minimal SMS configuration

Creating ACS routines

After you define the SMS classes and group, develop their respective ACS routines. For a minimal SMS configuration, in the *storage class ACS routine*, you assign a null storage class, as shown in the sample storage class ACS routine in Figure 5-21. The storage class ACS routine ensures that the storage class read/write variable is always set to null. This prevents users from externally specifying a storage class on their DD statements (STORCLAS keyword), which would cause the data set to be system-managed before you are ready.

The *storage group ACS routine* will never run if a null storage class is assigned. Therefore, no data sets are allocated as system-managed by the minimal configuration. However, you must code a trivial one to satisfy the SMS requirements for a valid SCDS. After you have written the ACS routines, use ISMF to translate them into executable form.

Follow these steps to create a data set that contains your ACS routines:

1. If you do not have the starter set, allocate a fixed-block PDS or PDSE with LRECL=80 to contain your ACS routines. Otherwise, start with the next step.
2. On the ISMF Primary Option Menu, select **Automatic Class Selection** to display the ACS Application Selection panel.
3. Select option **1 Edit**. When the next panel is shown, enter in the Edit panel the name of the PDS or PDSE data set you want to create to contain your source ACS routines.

Translating the ACS routines

The translation process checks the routines for syntax errors and converts the code into an ACS object. If the code translates without any syntax errors, then the ACS object is stored in the SCDS. For translate:

1. From the ISMF ACS Application Selection Menu panel, select **2 Translate**.
2. Enter your SCDS data set name, the PDS or PDSE data set name containing the ACS source routine, and a data set name to hold the translate output listing. When the listing data set does not exist, it is created automatically.

Validating the SCDS

When you validate your SCDS, you verify that all classes and groups assigned by your ACS routines are defined in the SCDS. To validate the SCDS:

1. From the ISMF Primary Option Menu panel, select **Control Data Set** and press Enter.
2. Enter your SCDS data set name and select **4 Validate.**

For more information, see *z/OS DFSMS: Using the Interactive Storage Management Facility,* SC26-7411.

5.22 DFSMS setup for z/OS

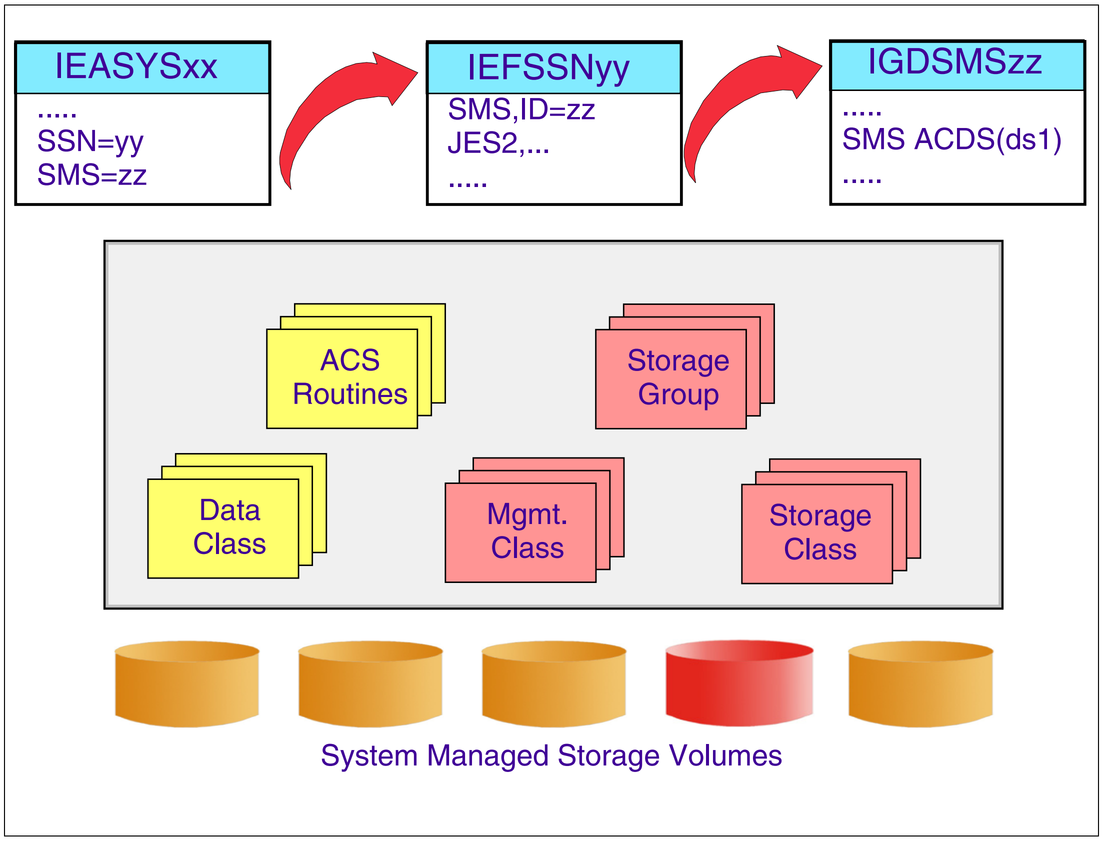


Figure 5-22 DFSMS setup for z/OS

DFSMS setup for z/OS

In preparation for starting SMS, update the following PARMLIB members to define SMS to z/OS:

IEASYSxx	Verify the suffix of the IEFSSNyy in use and add the SMS=xx parameter, where xx is the IGDSMS member name suffix.
IEFSSNyy	You can activate SMS only after you define the SMS subsystem to z/OS. To define SMS to z/OS, you must place a record for SMS in the IEFSSNxx PARMLIB member. IEFSSNxx defines how z/OS activates the SMS address space. You can code an IEFSSNxx member with keyword *or* positional parameters, but not both. We recommend using keyword parameters. We recommend that you place the SMS record *before* the JES2 record in IEFSSNxx in order to start SMS before starting the JES2 subsystem.
IGDSMSzz	For each system in the SMS complex, you must create an IGDSMSxx member in SYS1.PARMLIB. The IGDSMSzz member contains SMS initialization control information. The suffix has a default value of 00.

Every SMS system must have an IGDSMSzz member in SYS1.PARMLIB that specifies a required ACDS and COMMDS control data set pair. This ACDS and COMMDS pair is used if the COMMDS of the pair does not point to another COMMDS.

If the COMMDS points to another COMMDS, the referenced COMMDS is used. This referenced COMMDS might contain the name of an ACDS that is different from the one specified in the IGDSMSzz. If so, the name of the ACDS is obtained from the COMMDS rather than from the IGDSMSzz to ensure that the system is always running under the most recent ACDS and COMMDS.

If the COMMDS of the pair refers to another COMMDS during IPL, it means a more recent COMMDS has been used. SMS uses the most recent COMMDS to ensure that you cannot IPL with a down-level configuration.

The data sets that you specify for the ACDS and COMMDS pair must be the same for every system in an SMS complex. Whenever you change the ACDS or COMMDS, update the IGDSMSzz for every system in the SMS complex so that it specifies the same data sets.

IGDSMSzz has many parameters. For a complete description of the SMS parameters, see *z/OS MVS Initialization and Tuning Reference,* SA22-7592, and *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

5.23 Starting SMS and activating a new configuration

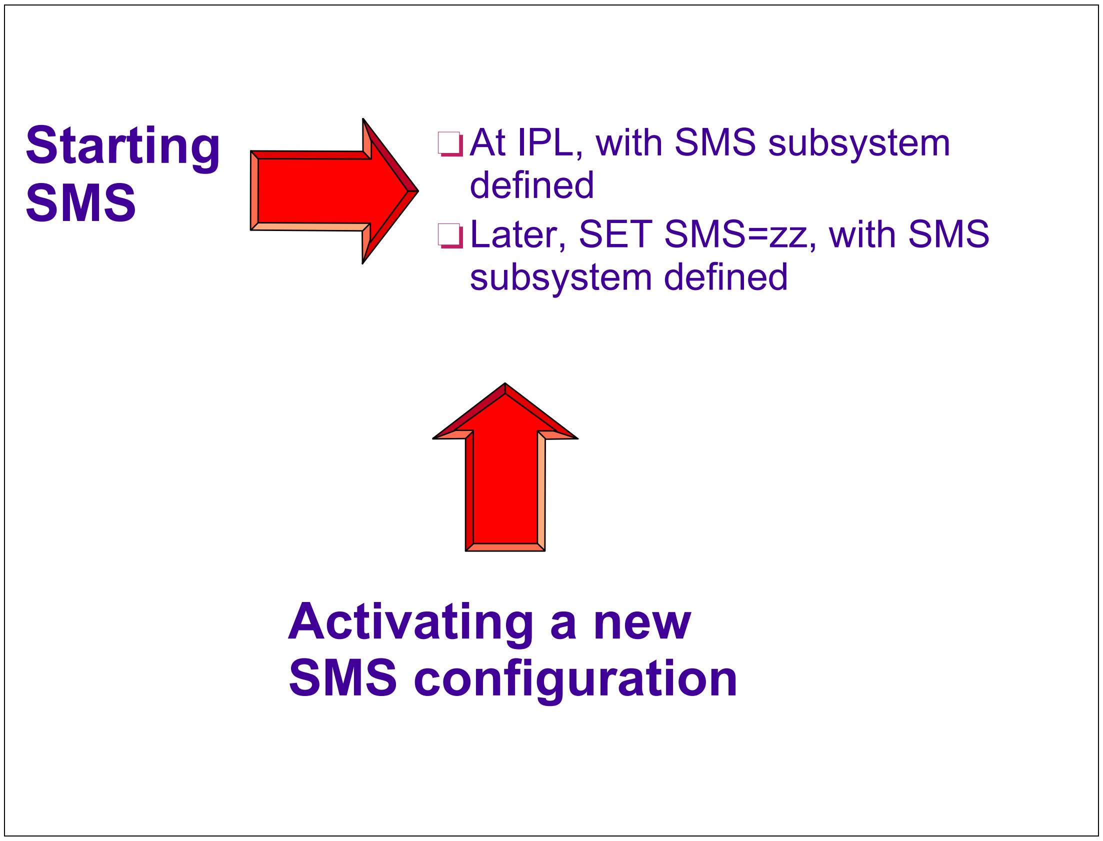

Starting SMS
At IPL, with SMS subsystem defined
Later, SET SMS=zz, with SMS subsystem defined
Activating a new SMS configuration

Figure 5-23 Starting SMS and activating a new SMS configuration

Starting SMS

To start SMS—which starts the SMS address space—use either of these methods:

- With SMS=xx defined in IEASYSxx and SMS defined as a valid subsystem, IPL the system. This starts SMS automatically.
- With SMS defined as a valid subsystem to z/OS, IPL the system. Start SMS later, using the `SET SMS=yy` MVS operator command.

For detailed information, refer to *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

Activating a new SMS configuration

Activating a new SMS configuration means to copy the configuration from SCDS to ACDS and to the SMS address space. The SCDS itself is never considered active. Attempting to activate an ACDS that is not valid results in an error message.

You can manually activate a new SMS configuration in two ways. Note that SMS must be active before you use one of these methods:

1. Activating an SMS configuration from ISMF:
   - From the ISMF Primary Option Menu panel, select **Control Data Set**.
   - In the CDS Application Selection panel, enter your SCDS data set name and select **5 Activate**, or enter the `ACTIVATE` command on the command line.

2. Activating an SMS configuration from the operator console:
   - From the operator console, enter the command:

     ```
     SETSMS {ACDS(YOUR.OWN.ACDS)} {SCDS(YOUR.OWN.SCDS)}
     ```

   Activating the configuration means that information is brought into the SMS address space from the ACDS.

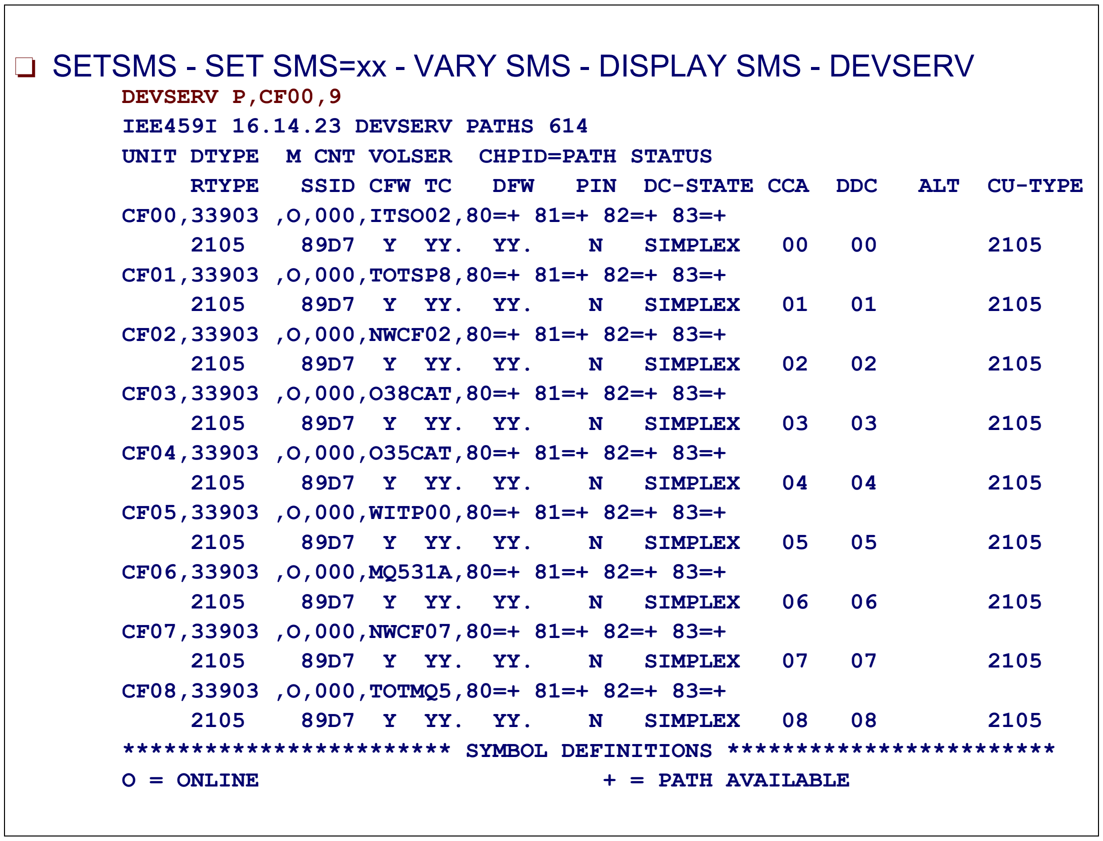
   To update the current ACDS with the contents of an SCDS, specify the SCDS parameter only.

   If you want to both specify a new ACDS and update it with the contents of an SCDS, enter the **SETSMS** command with both the ACDS and SCDS parameters specified.

The **ACTIVATE** command, which runs from the ISMF CDS application, is equivalent to the **SETSMS** operator command with the SCDS keyword specified.

If you use RACF, you can enable storage administrators to activate SMS configurations from ISMF by defining the facility STGADMIN.IGD.ACTIVATE.CONFIGURATION and issuing permit commands for each storage administrator.

5.24 Control SMS processing with operator commands



❑ SETSMS - SET SMS=xx - VARY SMS - DISPLAY SMS - DEVSERV

```
DEVSERV P,CF00,9
IEE459I 16.14.23 DEVSERV PATHS 614
UNIT DTYPE  M CNT VOLSER  CHPID=PATH STATUS
     RTYPE   SSID CFW TC   DFW   PIN  DC-STATE CCA  DDC   ALT  CU-TYPE
CF00,33903 ,O,000,ITSO02,80=+ 81=+ 82=+ 83=+
     2105    89D7  Y  YY.  YY.    N   SIMPLEX   00   00        2105
CF01,33903 ,O,000,TOTSP8,80=+ 81=+ 82=+ 83=+
     2105    89D7  Y  YY.  YY.    N   SIMPLEX   01   01        2105
CF02,33903 ,O,000,NWCF02,80=+ 81=+ 82=+ 83=+
     2105    89D7  Y  YY.  YY.    N   SIMPLEX   02   02        2105
CF03,33903 ,O,000,O38CAT,80=+ 81=+ 82=+ 83=+
     2105    89D7  Y  YY.  YY.    N   SIMPLEX   03   03        2105
CF04,33903 ,O,000,O35CAT,80=+ 81=+ 82=+ 83=+
     2105    89D7  Y  YY.  YY.    N   SIMPLEX   04   04        2105
CF05,33903 ,O,000,WITP00,80=+ 81=+ 82=+ 83=+
     2105    89D7  Y  YY.  YY.    N   SIMPLEX   05   05        2105
CF06,33903 ,O,000,MQ531A,80=+ 81=+ 82=+ 83=+
     2105    89D7  Y  YY.  YY.    N   SIMPLEX   06   06        2105
CF07,33903 ,O,000,NWCF07,80=+ 81=+ 82=+ 83=+
     2105    89D7  Y  YY.  YY.    N   SIMPLEX   07   07        2105
CF08,33903 ,O,000,TOTMQ5,80=+ 81=+ 82=+ 83=+
     2105    89D7  Y  YY.  YY.    N   SIMPLEX   08   08        2105
************************ SYMBOL DEFINITIONS ************************
O = ONLINE                         + = PATH AVAILABLE
```

Figure 5-24 SMS operator commands

Controlling SMS processing using operator commands

The DFSMS environment provides a set of z/OS operator commands to control SMS processing. The **VARY**, **DISPLAY**, **DEVSERV**, and **SET** commands are MVS operator commands that support SMS operation.

SETSMS This command changes a subset of SMS parameters from the operator console without changing the active IGDSMSxx PARMLIB member. For example, you can use this command to activate a new configuration from an SCDS. The MVS operator must use **SETSMS** to recover from ACDS and COMMDS failures.

For an explanation about how to recover from ACDS and COMMDS failures, refer to *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

SET SMS=zz This command starts SMS, if it has not already been started, and is defined as a valid MVS subsystem. The command also:

- Changes options set on the IGDSMSxx PARMLIB member
- Restarts SMS if it has terminated
- Updates the SMS configuration

Table 5-1 on page 263 lists the differences between the **SETSMS** and **SET SMS** commands.

VARY SMS	This command changes storage group, volume, library, or drive status. You can use this command to: – Limit new allocations to a volume or storage group – Enable a newly-installed volume for allocations
DISPLAY SMS	Refer to "Displaying the SMS configuration" on page 264.
DEVSERV	This command displays information for a device. Use it to display the status of extended functions in operation for a given volume that is attached to a cache-capable 3990 storage control. An example of the output of this command is shown in Figure 5-24 on page 262.

Table 5-1 Comparison of SETSMS and SET SMS commands

Difference	**SET SMS=xx**	**SETSMS**
When and how to use the command.	Initializes SMS parameters and starts SMS if SMS is defined but not started at IPL. Changes SMS parameters when SMS is running.	Changes SMS parameters only when SMS is running.
Where the parameters are entered.	IGDSMSxx PARMLIB member.	At the console.
What default values are available.	Default values are used for non-specified parameters.	No default values. Parameters non-specified remain unchanged.

For more information about operator commands, refer to *z/OS MVS System Commands,* SA22-7627.

5.25 Displaying the SMS configuration

```
D SMS,SG(STRIPE),LISTVOL
IGD002I 16:02:30 DISPLAY SMS 581

STORGRP  TYPE     SYSTEM= 1 2 3 4
STRIPE   POOL             + + + +

VOLUME   UNIT     SYSTEM= 1 2 3 4                         STORGRP NAME
MHLV11                    D D D D                           STRIPE
MHLV12                    D D D D                           STRIPE
MHLV13                    D D D D                           STRIPE
MHLV14                    D D D D                           STRIPE
MHLV15                    + + + +                           STRIPE
SBOX28   6312             + + + +                           STRIPE
SBOX29   6412             + + + +                           STRIPE
SBOX3K   6010             + + + +                           STRIPE
SBOX3L   6108             + + + +                           STRIPE
SBOX3M   6204             + + + +                           STRIPE
SBOX3N   6309             + + + +                           STRIPE
SBOX30   6512             + + + +                           STRIPE
SBOX31   6013             + + + +                           STRIPE
***************************** LEGEND *****************************
. THE STORAGE GROUP OR VOLUME IS NOT DEFINED TO THE SYSTEM
+ THE STORAGE GROUP OR VOLUME IS ENABLED
- THE STORAGE GROUP OR VOLUME IS DISABLED
* THE STORAGE GROUP OR VOLUME IS QUIESCED
D THE STORAGE GROUP OR VOLUME IS DISABLED FOR NEW ALLOCATIONS ONLY
Q THE STORAGE GROUP OR VOLUME IS QUIESCED FOR NEW ALLOCATIONS ONLY
> THE VOLSER IN UCB IS DIFFERENT FROM THE VOLSER IN CONFIGURATION
SYSTEM  1 = SC63       SYSTEM  2 = SC64       SYSTEM  3 = SC65
SYSTEM  4 = SC70
```

Figure 5-25 Display SMS configuration

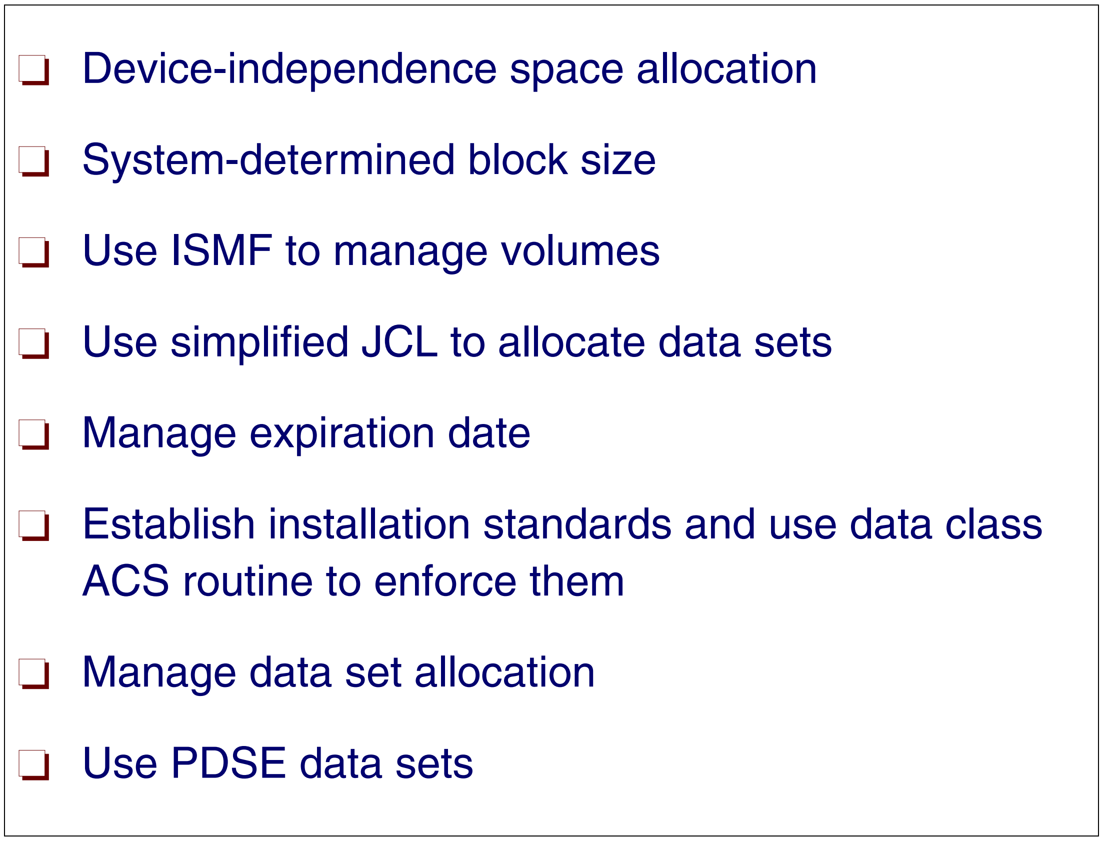
Displaying SMS configuration

You can display the SMS configuration in two ways:

- Using ISMF **Control Data Set**, enter `ACTIVE` in the CDS Name field and select **1 Display**.
- The **`DISPLAY SMS`** operator command shows volumes, storage groups, libraries, drives, SMS configuration information, SMS trace parameters, SMS operational options, OAM information, OSMC information, and cache information. Enter this command to:
  - Confirm that the system-managed volume status is correct
  - Confirm that SMS starts with the proper parameters

The **`DISPLAY SMS`** command can be used in different variations. For the full functionality of this command, refer to *z/OS MVS System Commands,* SA22-7627.

5.26 Managing data with a minimal SMS configuration



- Device-independence space allocation
- System-determined block size
- Use ISMF to manage volumes
- Use simplified JCL to allocate data sets
- Manage expiration date
- Establish installation standards and use data class ACS routine to enforce them
- Manage data set allocation
- Use PDSE data sets

Figure 5-26 Managing data with minimal SMS configuration

Managing data allocation

After the SMS minimal configuration is active, your installation can exploit some SMS capabilities that give you experience with SMS and help you plan the DFSMS full exploitation with system-managed data set implementation.

Inefficient space usage and poor data allocation cause problems with space and performance management. In a DFSMS environment, you can enforce good allocation practices to help reduce some of these problems. The following section highlights how to exploit SMS capabilities.

Using data classes to standardize data allocation

You can define data classes containing standard data set allocation attributes. Users then only need to use the appropriate data class names to create standardized data sets. To override values in the data class definition, they can still provide specific allocation parameters.

Data classes can be determined from the user-specified value on the DATACLAS parameter (DD card, TSO Alloc, Dynalloc macro), from a RACF default, or by ACS routines. ACS routines can also override user-specified or RACF default data classes.

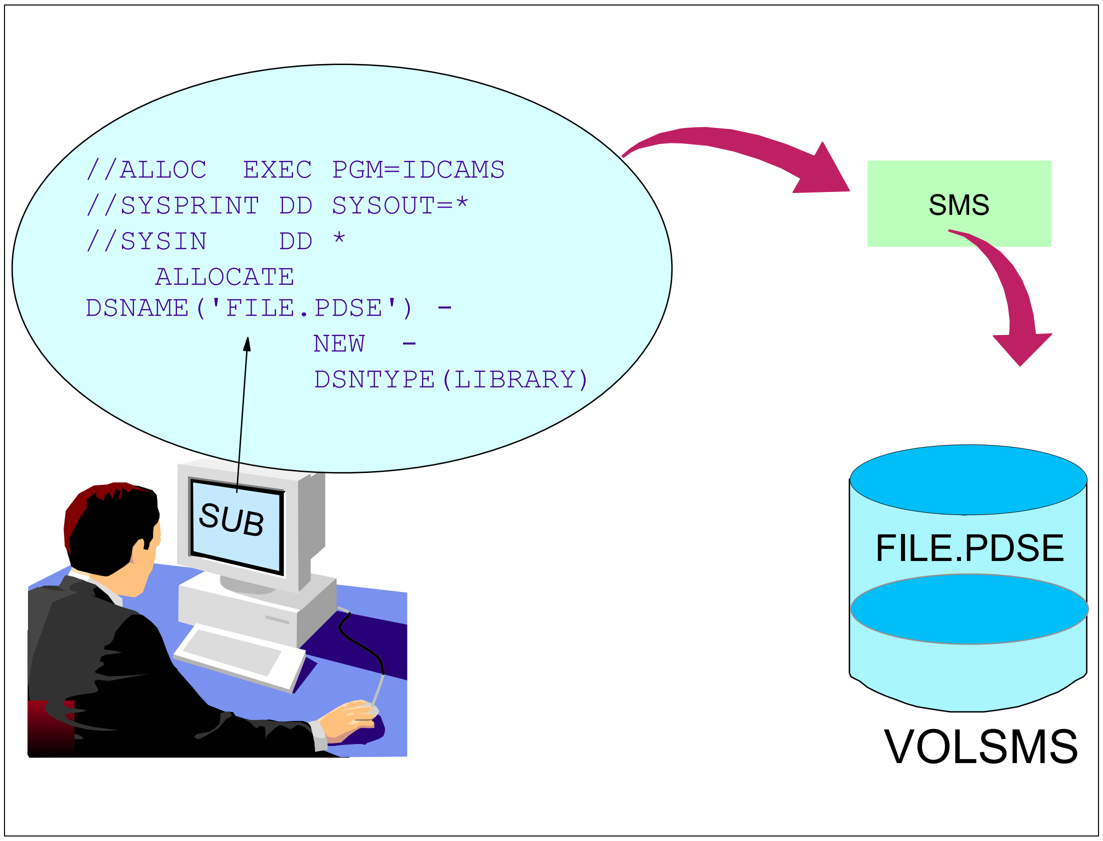
You can override a data class attribute (not the data class itself) using JCL or dynamic allocation parameters. DFSMS usually does not change values that are explicitly specified, because doing so would alter the original meaning and intent of the allocation. There is an exception. If it is clear that a PDS is being allocated (DSORG=PO or DSNTYPE=PDS is specified), and no directory space is indicated in the JCL, then the directory space from the data class is used.

Users cannot override the data class attributes of dynamically-allocated data sets if you use the IEFDB401 user exit.

For additional information about data classes see also “Using data classes” on page 231.

For sample data classes, descriptions, and ACS routines, see *z/OS DFSMS Implementing System-Managed Storage,* SC26-7407.

5.27 Device-independence space allocation



```
//ALLOC   EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN     DD *
    ALLOCATE
DSNAME('FILE.PDSE') -
             NEW   -
             DSNTYPE(LIBRARY)
```

Figure 5-27 Device independence

Ensuring device independence

The base configuration contains a default unit that corresponds to a DASD esoteric (such as SYSDA). Default geometry for this unit is specified in bytes/track and tracks/cylinder for the predominant device type in the esoteric. If users specify the esoteric, or do not supply the UNIT parameter for new allocations, the default geometry converts space allocation requests into device-independent units, such as KBs and MBs. This quantity is then converted back into device-dependent tracks based on the default geometry.

System-determined block size

During allocation, DFSMSdfp assists you to assign a block size that is optimal for the device. When you allow DFSMSdfp to calculate the block size for the data set, you are using a system-determined block size. System-determined block sizes can be calculated for system-managed and non-system-managed primary storage, VIO, and tape data sets.

The use of system-determined block size provides:

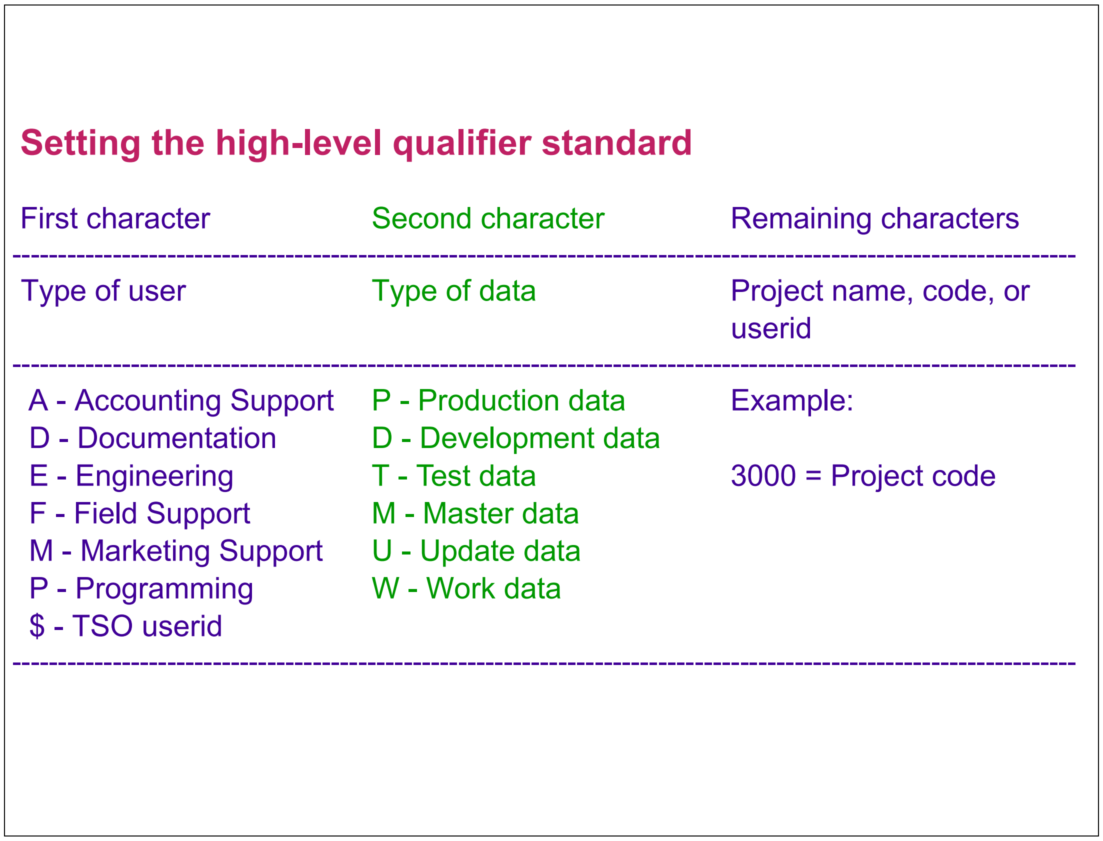
- Device independence, since you do not need to know the track capacity to allocate efficiently
- Space usage optimization
- I/O performance improvement
- Simplifies JCL, since you do not need to code BLKSIZE

You take full advantage of system-managed storage when you allow the system to place data on the most appropriate device in the most efficient way, when you use *system-managed data* *sets* (see Figure 5-27). In the DFSMS environment, you control volume selection through the storage class and storage group definitions you create, and by ACS routines. This means that users do not have to specify volume serial numbers with the VOL=SER parameter, or code a specific device type with the UNIT= parameter on their JCL. Due to a large number of volumes in one installation, the volume selection SMS routine may take a long time. A fast volume selection routine, which improves the best-fit and not-best-fit algorithms, is introduced in z/OS 1.8.

When converting data sets for use in DFSMS, users do not have to remove these parameters from existing JCL because volume and unit information can be ignored with ACS routines. (However, you should work with users to evaluate UNIT and VOL=SER dependencies before conversion).

If you keep the VOL=SER parameter for a non-SMS volume, but you are trying to access a system-managed data set, then SMS might not find the data set. All SMS data sets (the ones with a storage class) must reside on a *system-managed volume*.

5.28 Developing naming conventions



Setting the high-level qualifier standard

First character	Second character	Remaining characters
Type of user	Type of data	Project name, code, or userid
A - Accounting Support D - Documentation E - Engineering F - Field Support M - Marketing Support P - Programming $ - TSO userid	P - Production data D - Development data T - Test data M - Master data U - Update data W - Work data	Example: 3000 = Project code

Figure 5-28 Setting data set HLQ conventions

Developing a data set naming convention

Whenever you allocate a new data set, you (or the operating system) must give the data set a unique name. Usually, the data set name is given as the dsname in JCL. A data set name can be one name segment, or a series of joined name segments. See also 2.2, “Data set name rules” on page 19.

You must implement a naming convention for your data sets. Although a naming convention is not a prerequisite for DFSMS conversion, it makes more efficient use of DFSMS. You can also reduce the cost of storage management significantly by grouping data that shares common management requirements. Naming conventions are an effective way of grouping data. They also:

- Simplify service-level assignments to data
- Facilitate writing and maintaining ACS routines
- Allow data to be mixed in a system-managed environment while retaining separate management criteria
- Provide a filtering technique useful with many storage management products
- Simplify the data definition step of aggregate backup and recovery support

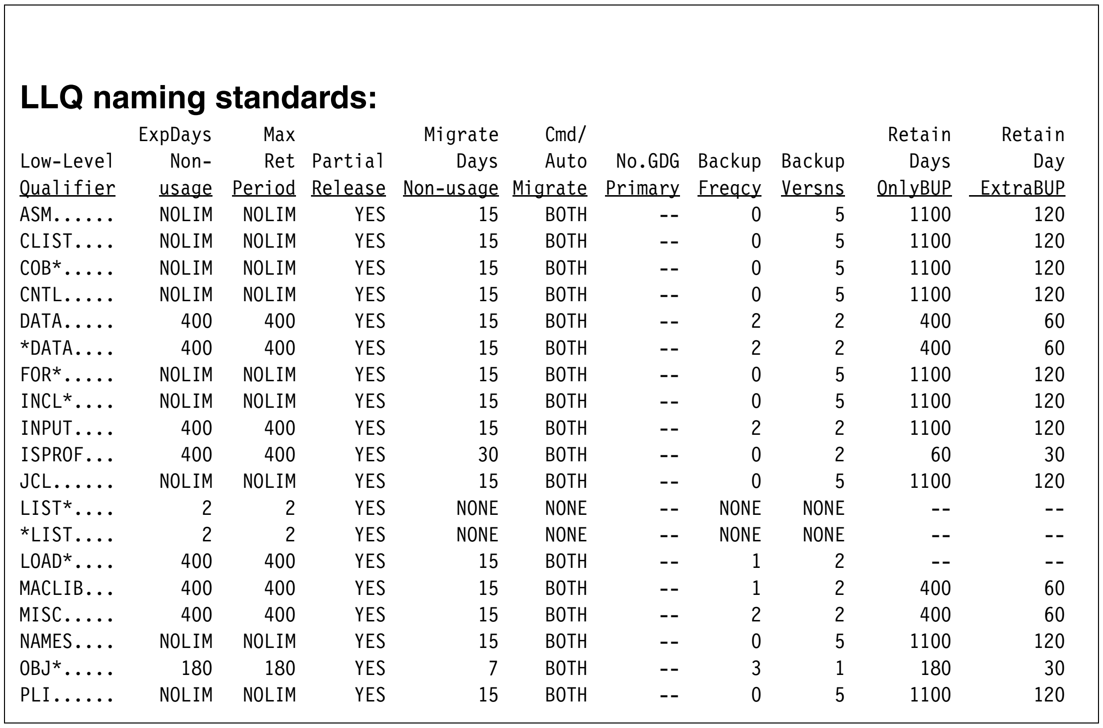
Most naming conventions are based on the HLQ and LLQ of the data name. Other levels of qualifiers can be used to identify generation data sets and database data. They can also be used to help users to identify their own data.

Using a high-level qualifier (HLQ)

Use the HLQ to identify the owner or owning group of the data, or to indicate data type.

Do not embed information that is subject to frequent change in the HLQ, such as department number, application location, output device type, job name, or access method. Set a standard within the HLQ. Figure 5-28 shows examples of naming standards.

5.29 Setting the low-level qualifier (LLQ) standards



LLQ naming standards:

Low-Level Qualifier	ExpDays Non-usage	Max Ret Period	Partial Release	Migrate Days Non-usage	Cmd/ Auto Migrate	No.GDG Primary	Backup Freqcy	Backup Versns	Retain Days OnlyBUP	Retain Day ExtraBUP
ASM......	NOLIM	NOLIM	YES	15	BOTH	--	0	5	1100	120
CLIST....	NOLIM	NOLIM	YES	15	BOTH	--	0	5	1100	120
COB*.....	NOLIM	NOLIM	YES	15	BOTH	--	0	5	1100	120
CNTL.....	NOLIM	NOLIM	YES	15	BOTH	--	0	5	1100	120
DATA.....	400	400	YES	15	BOTH	--	2	2	400	60
*DATA....	400	400	YES	15	BOTH	--	2	2	400	60
FOR*.....	NOLIM	NOLIM	YES	15	BOTH	--	0	5	1100	120
INCL*....	NOLIM	NOLIM	YES	15	BOTH	--	0	5	1100	120
INPUT....	400	400	YES	15	BOTH	--	2	2	1100	120
ISPROF...	400	400	YES	30	BOTH	--	0	2	60	30
JCL......	NOLIM	NOLIM	YES	15	BOTH	--	0	5	1100	120
LIST*....	2	2	YES	NONE	NONE	--	NONE	NONE	--	--
*LIST....	2	2	YES	NONE	NONE	--	NONE	NONE	--	--
LOAD*....	400	400	YES	15	BOTH	--	1	2	--	--
MACLIB...	400	400	YES	15	BOTH	--	1	2	400	60
MISC.....	400	400	YES	15	BOTH	--	2	2	400	60
NAMES....	NOLIM	NOLIM	YES	15	BOTH	--	0	5	1100	120
OBJ*.....	180	180	YES	7	BOTH	--	3	1	180	30
PLI......	NOLIM	NOLIM	YES	15	BOTH	--	0	5	1100	120

Figure 5-29 Setting the LLQ standards

Setting the low-level qualifier (LLQ) standards

The LLQ determines the contents and storage management processing of the data. You can use LLQs to identify data requirements for:

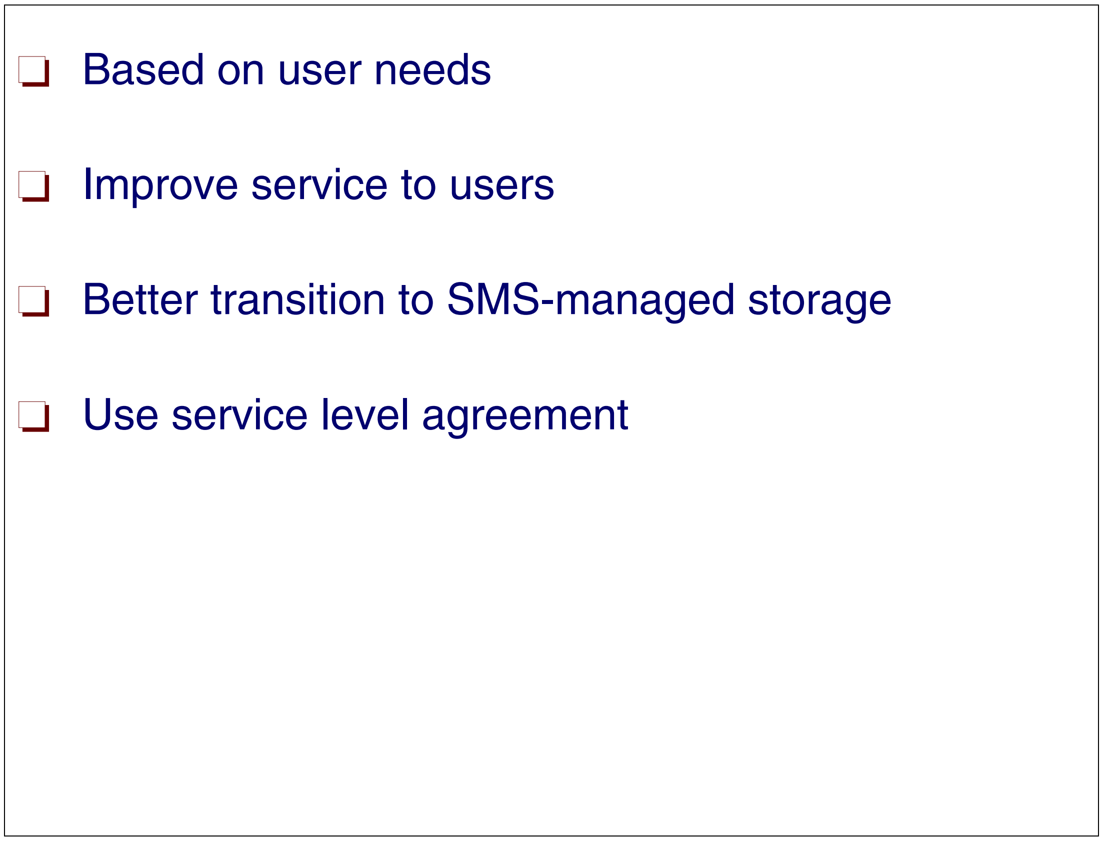
- Migration (data sets only)
- Backup (data sets and objects)
- Archiving (data sets)
- Retention or expiration (data sets and objects)
- Class transitions (objects only)
- Release of unused space (data sets only)

Mapping storage management requirements to data names is especially useful in a system-managed environment. In an environment without storage groups, data with differing requirements is often segregated onto separate volumes that are monitored and managed manually. LLQ data naming conventions allow data to be mixed together in a system-managed environment and still retain the separate management criteria.

Figure 5-29 shows examples of how you can use LLQ naming standards to indicate the storage management processing criteria.

The first column lists the LLQ of a data name. An asterisk indicates where a partial qualifier can be used. For example, LIST* indicates that only the first four characters of the LLQ must be LIST; valid qualifiers include LIST1, LISTING, and LISTOUT. The remaining columns show the storage management processing information for the data listed.

5.30 Establishing installation standards



❑ Based on user needs

❑ Improve service to users

❑ Better transition to SMS-managed storage

❑ Use service level agreement

Figure 5-30 Establishing installation standards

Establishing installation standards

Establishing standards such as naming conventions and allocation policies helps you to manage storage more efficiently and improves service to your users. With them, your installation is better prepared to make a smooth transition to system-managed storage.

Negotiate with your user group representatives to agree on the specific policies for the installation, how soon you can implement them, and how strongly you enforce them.

You can simplify storage management by limiting the number of data sets and volumes that cannot be system-managed.

Service level agreement

Document negotiated policies in a service level agreement. We recommend that you develop the following documents before you start to migrate permanent data to system management:

- A storage management plan that documents your storage administration group's strategy for meeting storage requirements
- Formal service level agreements that describe the services that you agree to provide

5.31 Planning and defining data classes

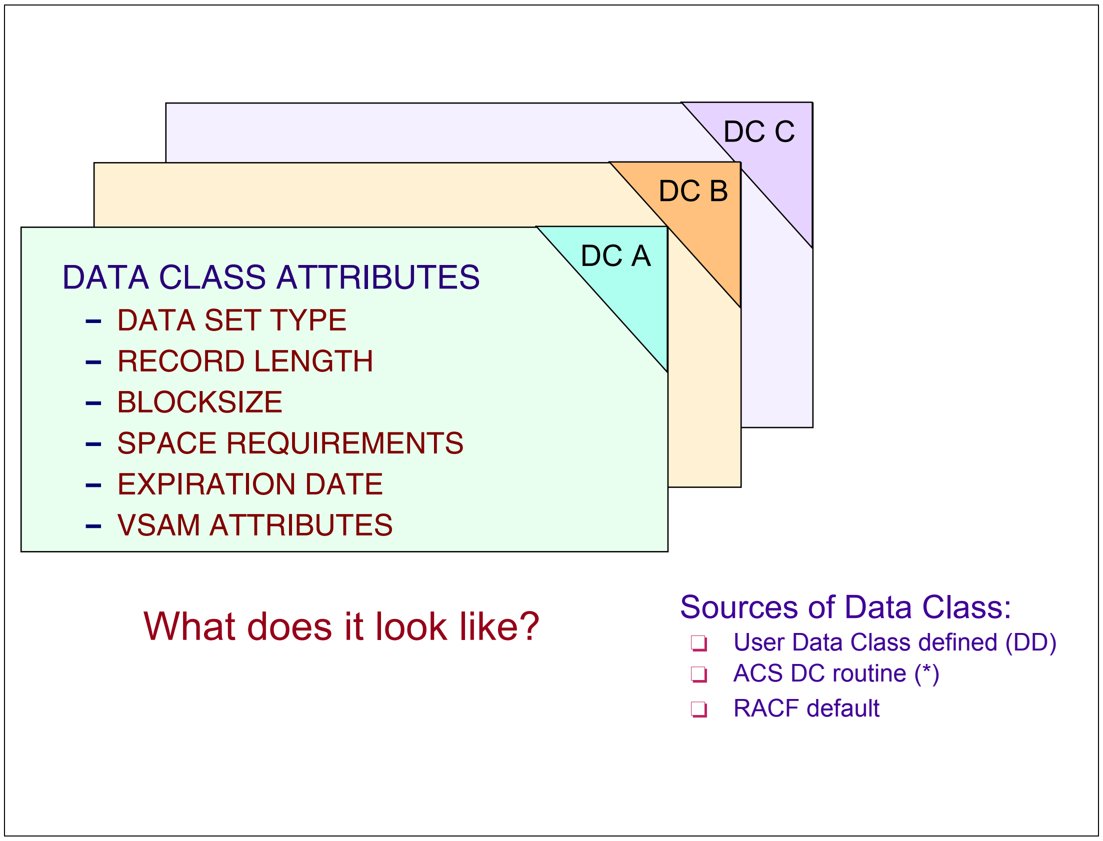


Figure 5-31 Planning and defining data class

Planning and defining data classes

After you establish your installation’s standards, use your service level agreement (SLA) for reference when planning your data classes. SLAs identify users’ current allocation practices and their requirements. For example:

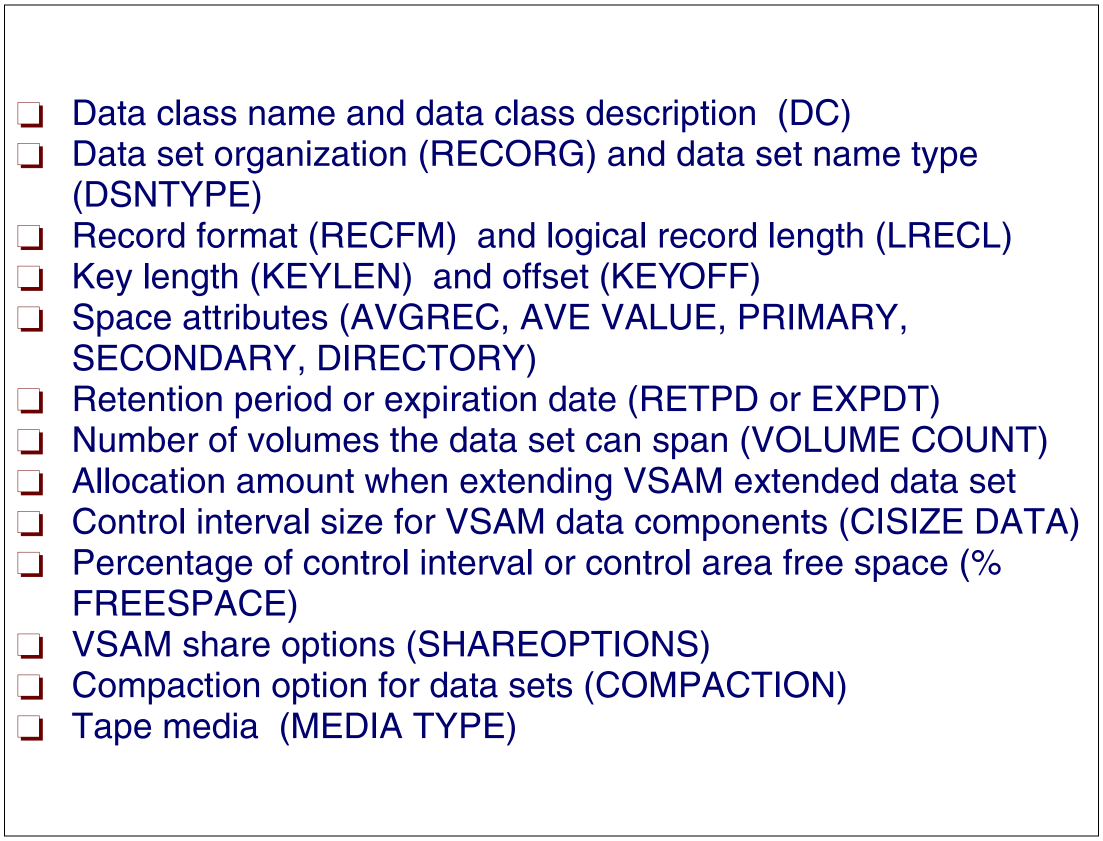
- Based on user requirements, you might create a data class to allocate standard control libraries.
- You can create a data class to supply the default value of a parameter, so users do not have to specify a value for that parameter in the JCL or dynamic allocation.

Data class names should indicate the type of data they are assigned to. This makes it easier for users to identify the template they need to use for allocation.

You define data classes using the ISMF data class application. Users can access the Data Class List panel to determine which data classes are available and the allocation values that each data class contains.

Figure 5-32 on page 274 contains information that can help in this task. For more information about planning and defining data classes, see *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

5.32 Data class attributes



- ❑ Data class name and data class description (DC)
- ❑ Data set organization (RECORG) and data set name type (DSNTYPE)
- ❑ Record format (RECFM) and logical record length (LRECL)
- ❑ Key length (KEYLEN) and offset (KEYOFF)
- ❑ Space attributes (AVGREC, AVE VALUE, PRIMARY, SECONDARY, DIRECTORY)
- ❑ Retention period or expiration date (RETPD or EXPDT)
- ❑ Number of volumes the data set can span (VOLUME COUNT)
- ❑ Allocation amount when extending VSAM extended data set
- ❑ Control interval size for VSAM data components (CISIZE DATA)
- ❑ Percentage of control interval or control area free space (% FREESPACE)
- ❑ VSAM share options (SHAREOPTIONS)
- ❑ Compaction option for data sets (COMPACTION)
- ❑ Tape media (MEDIA TYPE)

Figure 5-32 Data class attributes

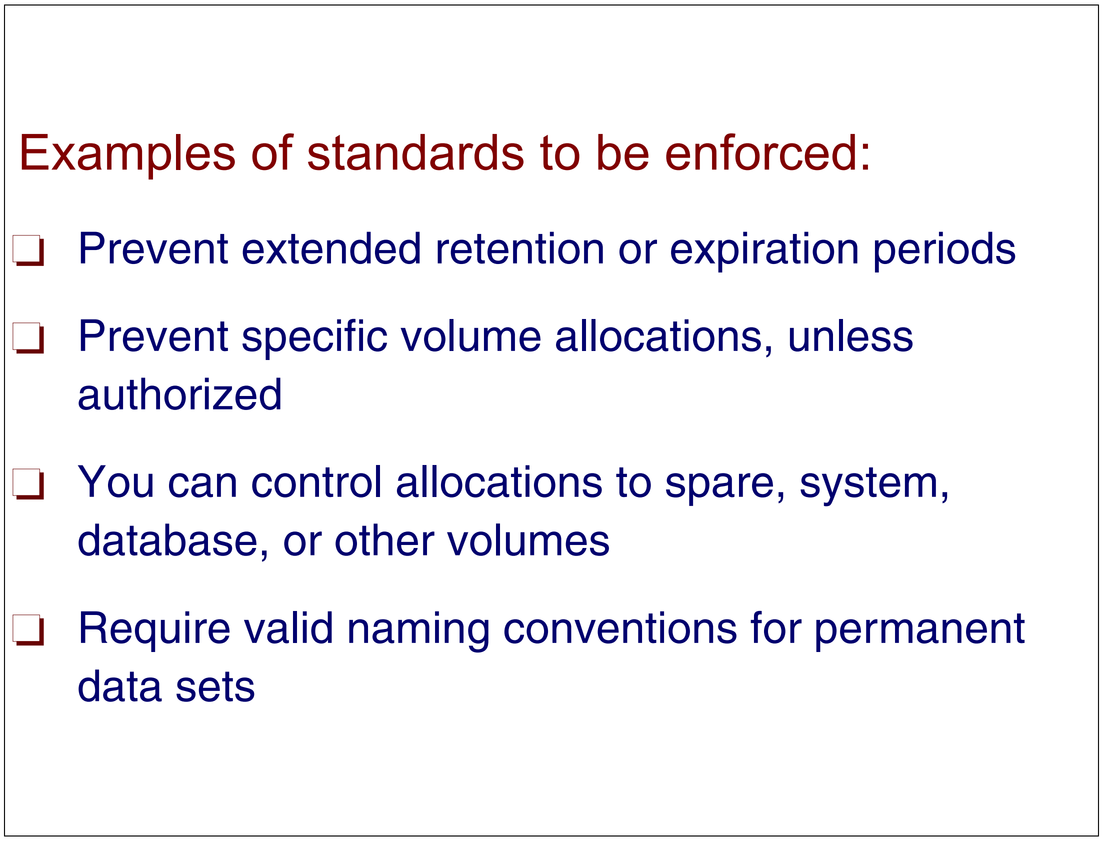
Data class attributes

You can specify the data class space attributes to control DASD space waste. For example:

- The primary space value should specify the total amount of space initially required for output processing. The secondary allocation allows automatic extension of additional space as the data set grows and does not waste space by overallocating the primary quantity. You can also use data class space attributes to relieve users of the burden of calculating how much primary and secondary space to allocate.
- The COMPACTION attribute specifies whether data is to be compressed on DASD if the data set is allocated in the extended format. The COMPACTION attribute alone also allows you to use the improved data recording capability (IDRC) of your tape device when allocating tape data sets. To use the COMPACTION attribute, the data set *must* be system-managed, since this attribute demands an extended format data set.
- The following attributes are used for tape data sets only:
  - MEDIA TYPE allows you to select the mountable tape media cartridge type.
  - RECORDING TECHNOLOGY allows you to select the format to use when writing to that device.
  - The read-compatible special attribute indicator in the tape device selection information (TDSI) allows an 18-track tape to be mounted on a 36-track device for read access. The attribute increases the number of devices that are eligible for allocation when you are certain that no more data will be written to the tape.

For detailed information about specifying data class attributes, see *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

5.33 Use data class ACS routine to enforce standards



Examples of standards to be enforced:

- ❑ Prevent extended retention or expiration periods
- ❑ Prevent specific volume allocations, unless authorized
- ❑ You can control allocations to spare, system, database, or other volumes
- ❑ Require valid naming conventions for permanent data sets

Figure 5-33 Using data class (DC) ACS routine to enforce standards

Using data class (DC) ACS routine to enforce standards

Once you started DFSMS with the minimal configuration, you can use data class ACS routine facilities to automate or simplify storage allocation standards if you:

- ► Use manual techniques to enforce standards
- ► Plan to enforce standards before implementing DFSMS
- ► Use DFSMSdfp or MVS installation exits to enforce storage allocation standards

The data class ACS routine provides an automatic method for enforcing standards because it is called for *system-managed* and *non-system-managed* data set allocations. Standards are enforced automatically at allocation time, rather than through manual techniques after allocation.

Enforcing standards optimizes data processing resources, improves service to users, and positions you for implementing system-managed storage. You can fail requests or issue warning messages to users who do not conform to standards. Consider enforcing the following standards in your DFSMS environment:

- ► Prevent extended retention or expiration periods.
- ► Prevent specific volume allocations, unless authorized. For example, you can control allocations to spare, system, database, or other volumes.

Require valid naming conventions before implementing DFSMS system management for permanent data sets.

5.34 Simplifying JCL use

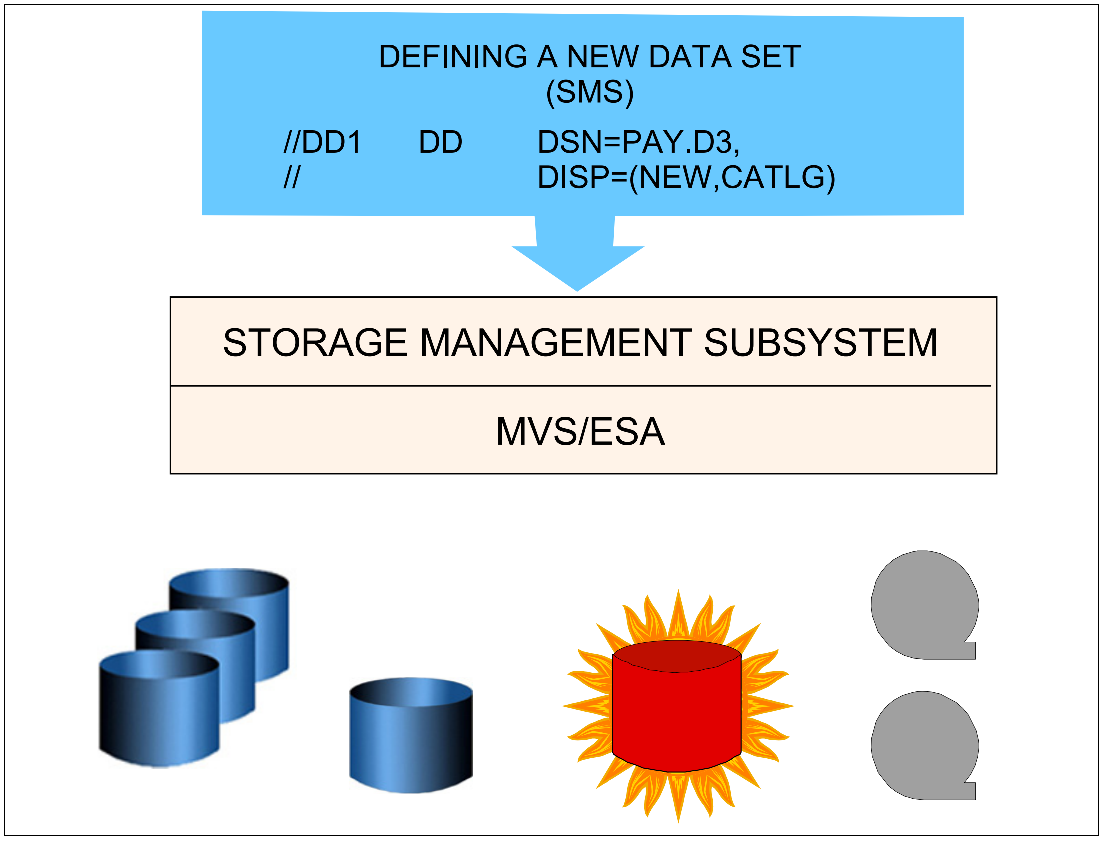


Figure 5-34 Using SMS capabilities to simplify JCL

Use simplified JCL

Once you have defined and start using data classes, several JCL keywords can help you simplify the task of creating data sets and also make the allocation process more consistent. It is also possible to allocate VSAM data sets through JCL without IDCAMS assistance.

For example, with the use of data classes, you have less use for the JCL keywords: UNIT, DCB, and AMP. When you start using *system-managed data sets*, you do not need to use the JCL VOL keyword.

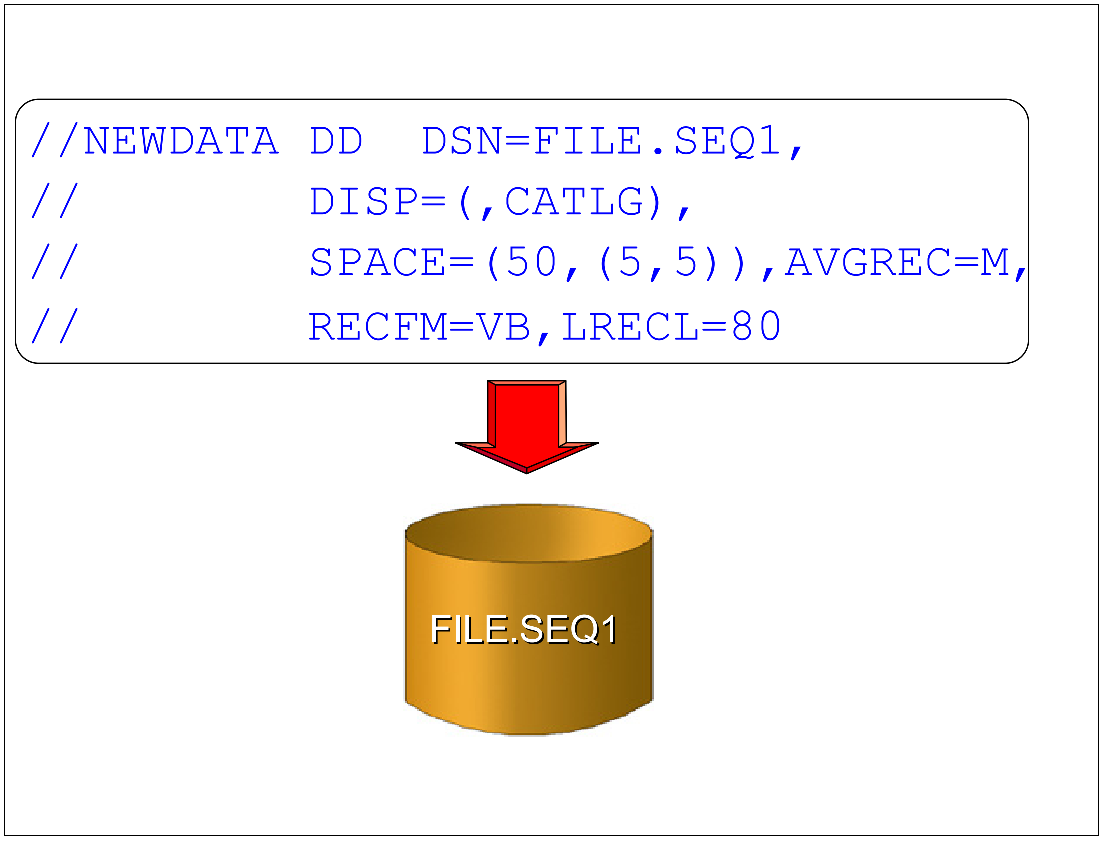
JCL keywords used in the DFSMS environment

You can use JCL keywords to create VSAM and non-VSAM data sets. For a detailed description of the keywords and their use, see *z/OS MVS JCL User's Guide,* SA22-7598.

In the following sections, we present some sample jobs exemplifying the use of JCL keywords when:

- Creating a sequential data set
- Creating a VSAM cluster
- Specifying a retention period
- Specifying an expiration date

5.35 Allocating a data set



```
//NEWDATA DD  DSN=FILE.SEQ1,
//            DISP=(,CATLG),
//            SPACE=(50,(5,5)),AVGREC=M,
//            RECFM=VB,LRECL=80
```

Figure 5-35 Allocating a sequential data set

Creating and allocating data sets

Many times the words *create* and *allocate*, when applied to data sets, are used in MVS as synonyms. However, they are not.

- To *create* (on DASD) means to assign a space in VTOC to be used for a data set (sometimes *create* implies cataloging the data set). A data set is created in response to the DD card DISP=NEW in JCL.
- To *allocate* means to establish a logical relationship between the request for the use of the data set within the program (through the use of a DCB or ACB) and the data set itself in the device where it is located.

Figure 5-35 shows an example of JCL used to create a data set in a system-managed environment.

These are some characteristics of the JCL in a system-managed environment:

- The LRECL and RECFM parameters are independent keywords. This makes it easier to override individual attributes that are assigned default values by the data class.
- In the example, the SPACE parameter is coded with the average number of bytes per record (50), and the number of records required for the primary data set allocation (5 M) and secondary data set allocation (5 M). These are the values that the system uses to calculate the least number of tracks required for the space allocation.

- The AVGREC attribute indicates the scale factor for the primary and secondary allocation values. In the example, an AVGREC value of M indicates that the primary and secondary values of 5 are each to be multiplied by 1 048 576.
- For *system-managed data sets*, the device-dependent volume serial number and unit information is no longer required, because the volume is assigned within a storage group selected by the ACS routines.

Overriding data class attributes with JCL

In a DFSMS environment, the JCL to allocate a data set is simpler and has no device-dependent keywords.

Table 5-2 lists the attributes a user can override with JCL.

Table 5-2 Data class attributes that can be overridden by JCL

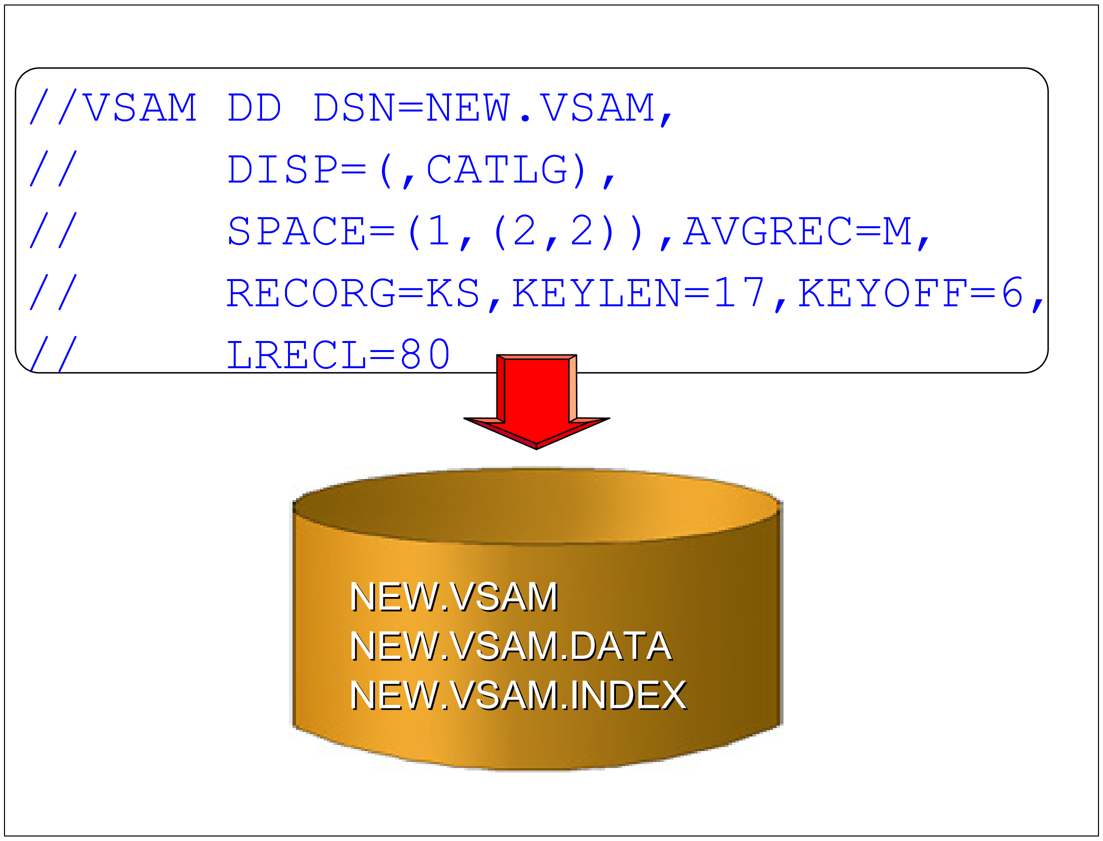
JCL DD statement keyword	Use for
RECORG,KEYLEN,KEYOFF	Only VSAM
RECFM	Sequential (PO or PS)
LRECL,SPACE,AVGREC, RETPD or EXPDT, VOLUME (volume count)	All data set types
DSNTYPE	PDS or PDSE

For more information about data classes refer to “Using data classes” on page 231 and “Data class attributes” on page 274.

As previously mentioned, in order to use a data class, the data set does *not* have to be system-managed. An installation can take advantages of a minimal SMS configuration to simplify JCL use and manage data set allocation.

For information about managing data allocation, refer to *z/OS DFSMS: Using Data Sets,* SC26-7410.

5.36 Creating a VSAM cluster



```
//VSAM DD DSN=NEW.VSAM,
//        DISP=(,CATLG),
//        SPACE=(1,(2,2)),AVGREC=M,
//        RECORG=KS,KEYLEN=17,KEYOFF=6,
//        LRECL=80
```

Figure 5-36 Creating a VSAM data class

Creating a VSAM data set using JCL

In the DFSMS environment, you can create temporary and permanent VSAM data sets using JCL by using either of the following:

- The RECORG parameter of the JCL DD statement
- A data class

You can use JCL DD statement parameters to override some data class attributes; refer to Table 5-2 for those related to VSAM data sets.

Attention: Regarding DISP=(OLD,DELETE), in an SMS environment, the VSAM data set is deleted at unallocation. In a non-SMS environment, the VSAM data set is kept.

A data set with a disposition of MOD is treated as a NEW allocation if it does not already exist; otherwise, it is treated as an OLD allocation.

For a non-SMS environment, a VSAM cluster creation is only done through IDCAMS. In Figure 5-36, NEW.VSAM refers to a KSDS VSAM cluster.

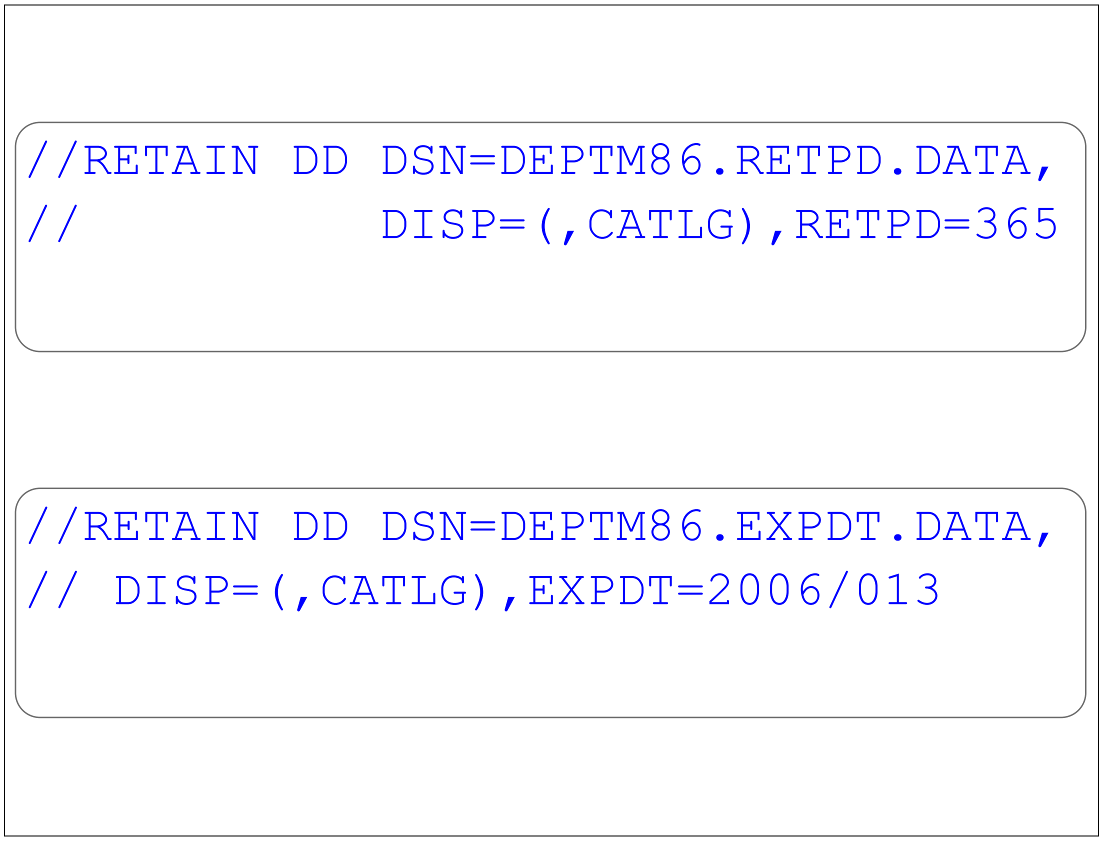
Considerations when specifying space for a KSDS

The space allocation for a VSAM entity depends on the level of the entity being allocated:

- If allocation is specified at the cluster or alternate index level only, the amount needed for the index is subtracted from the specified amount. The remainder of the specified amount is assigned to the data component.
- If allocation is specified at the data level only, the specified amount is assigned to data. The amount needed for the index is in addition to the specified amount.
- If allocation is specified at both the data and index levels, the specified data amount is assigned to data and the specified index amount is assigned to the index.
- If secondary allocation is specified at the data level, secondary allocation must be specified at the index level or the cluster level.

You cannot use certain parameters in JCL when allocating VSAM data sets, although you can use them in the IDCAMS **`DEFINE`** command.

5.37 Retention period and expiration date



```
//RETAIN DD DSN=DEPTM86.RETPD.DATA,
//          DISP=(,CATLG),RETPD=365
```

```
//RETAIN DD DSN=DEPTM86.EXPDT.DATA,
// DISP=(,CATLG),EXPDT=2006/013
```

Figure 5-37 Retention period and expiration date

Managing retention period and expiration date

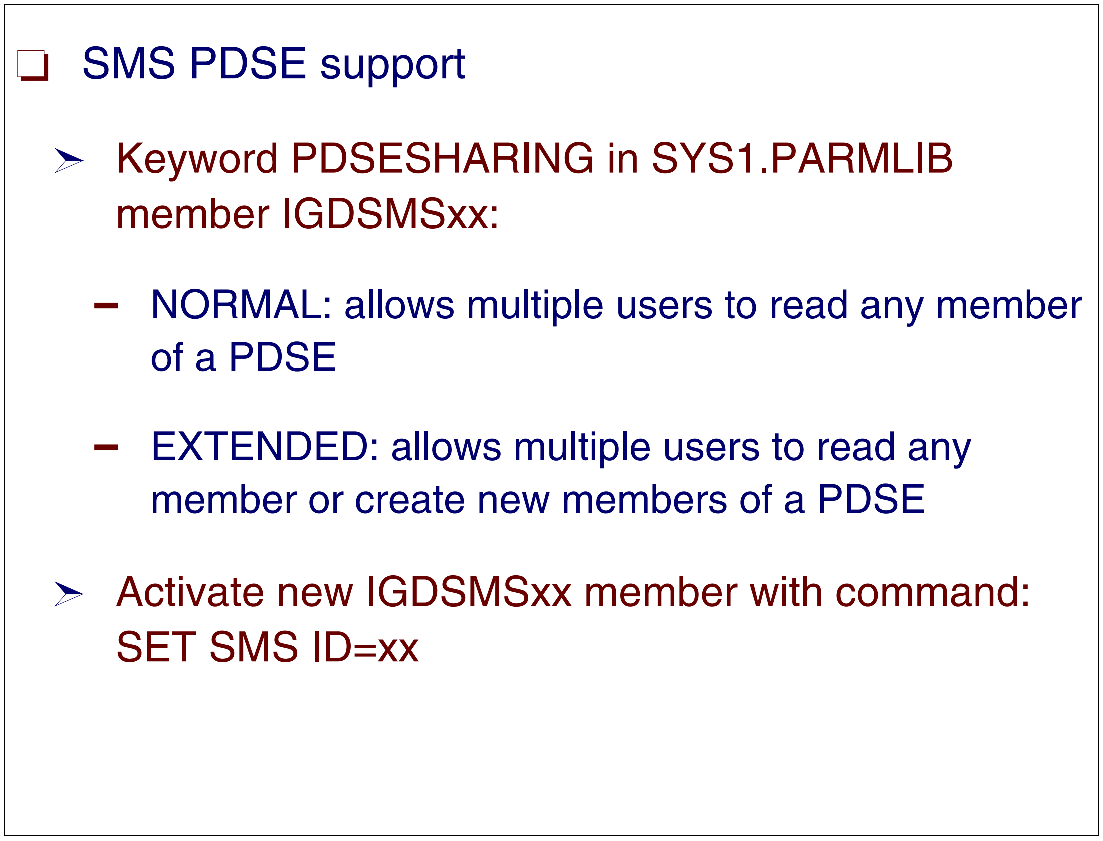
The RETPD and EXPDT parameters specify retention period and expiration date. They apply alike to system-managed and non-system-managed data sets. They control the time during which a data set is protected from being deleted by the system. The first DD statement in Figure 5-37 protects the data set from deletion *for* 365 days. The second DD statement in Figure 5-37 protects the data set from deletion *until* January 13, 2006.

The VTOC entry for non-VSAM and VSAM data sets contains the expiration date as declared in the JCL, the TSO **ALLOCATE** command, the IDCAMS **DEFINE** command, or in the data class definition. The expiration date is placed in the VTOC either directly from the date specification, or after it is calculated from the retention period specification. The expiration date in the catalog entry exists for information purposes only. If you specify the current date or an earlier date, the data set is immediately eligible for replacement.

You can use a *management class* to limit or ignore the RETPD and EXPDT parameters given by a user. If a user specifies values that exceed the maximum allowed by the management class definition, the retention period is reset to the allowed maximum. For an expiration date beyond year 1999 use the following format: YYYY/DDD. For more information about using management class to control retention period and expiration date, refer to *z/OS DFSMShsm Storage Administration Guide,* SC35-0421.

Attention: Expiration dates 99365, or 99366, or 1999/365 or 1999/366 are special dates and they mean *never expires*.

5.38 SMS PDSE support



- SMS PDSE support
  - Keyword PDSESHARING in SYS1.PARMLIB member IGDSMSxx:
    - NORMAL: allows multiple users to read any member of a PDSE
    - EXTENDED: allows multiple users to read any member or create new members of a PDSE
  - Activate new IGDSMSxx member with command: SET SMS ID=xx

Figure 5-38 SMS PDSE support

SMS PDSE support

With the minimal SMS configuration, you can exploit the use of PDSE data sets. A PDSE does not have to be system-managed. For information about PDSE, refer to “Partitioned data set extended (PDSE)” on page 151.

If you have DFSMS installed, you can extend PDSE sharing to enable multiple users on multiple systems to concurrently create new PDSE members and read existing members.

Using the PDSESHARING keyword in the SYS1.PARMLIB member, IGDSMSxx, you can specify:

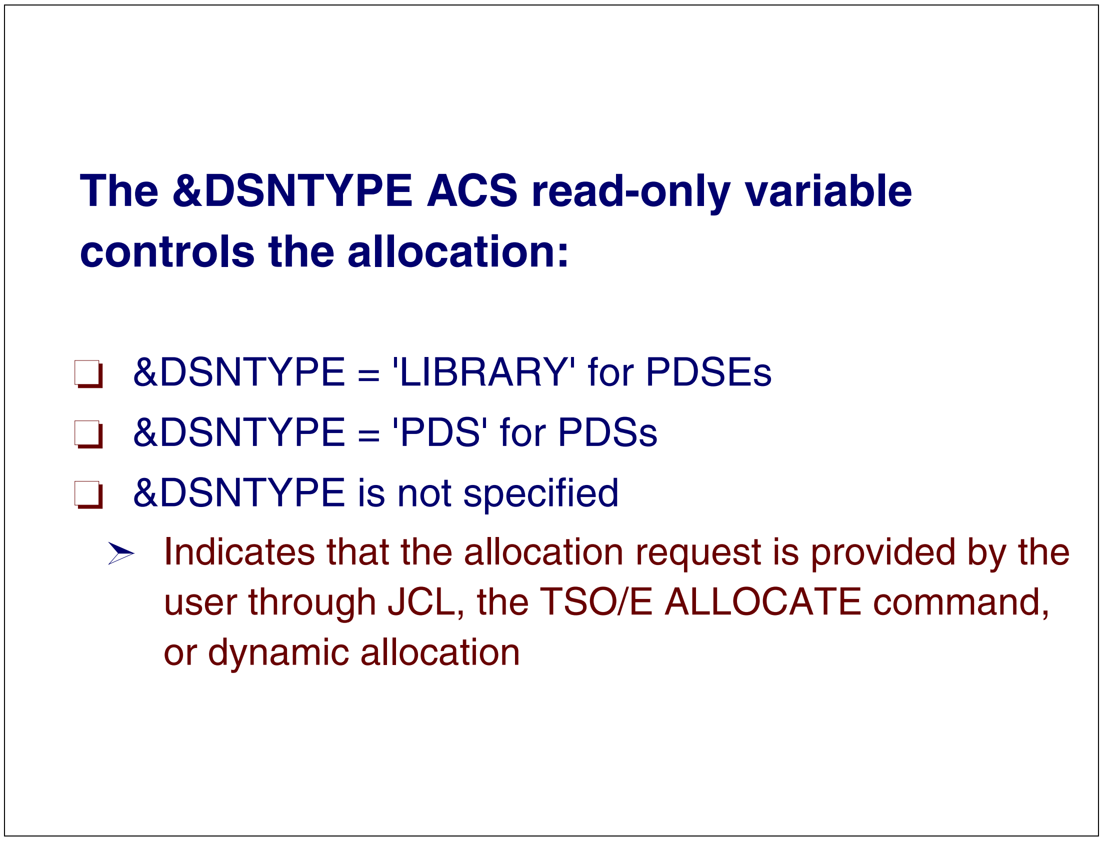
- NORMAL. This allows multiple users to read any member of a PDSE.
- EXTENDED. This allows multiple users to read any member or create new members of a PDSE.

All systems sharing PDSEs need to be upgraded to DFSMS to use the extended PDSE sharing capability.

After updating the IGDSMSxx member of SYS1.PARMLIB, you need to issue the **`SET SMS ID=xx`** command for every system in the complex to activate the sharing capability. See also *z/OS DFSMS: Using Data Sets,* SC26-7410 for information about PDSE sharing.

Although SMS supports PDSs, you should consider converting these to the PDSE format. Refer to 4.26, “PDSE: Conversion” on page 155 for more information about PDSE conversion.

5.39 Selecting data sets to allocate as PDSEs



The &DSNTYPE ACS read-only variable controls the allocation:

- &DSNTYPE = 'LIBRARY' for PDSEs
- &DSNTYPE = 'PDS' for PDSs
- &DSNTYPE is not specified
  - Indicates that the allocation request is provided by the user through JCL, the TSO/E ALLOCATE command, or dynamic allocation

Figure 5-39 Selecting a data set to allocate as PDSE

Selecting a data set to allocate as PDSE

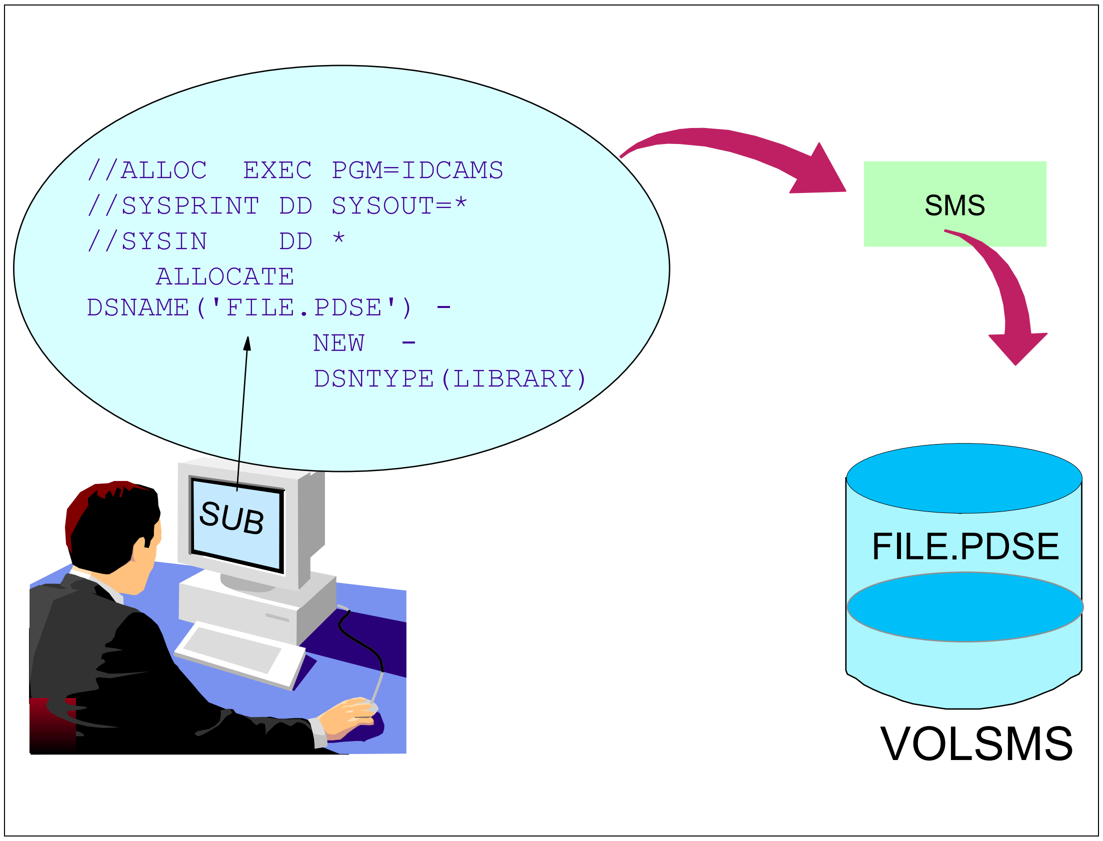
As a storage administrator, you can code appropriate ACS routines to select data sets to allocate as PDSEs and prevent inappropriate PDSs from being allocated or converted to PDSEs.

By using the &DSNTYPE read-only variable in the ACS routine for data-class selection, you can control which PDSs are to be allocated as PDSEs. The following values are valid for DSNTYPE in the data class ACS routines:

- &DSNTYPE = 'LIBRARY' for PDSEs.
- &DSNTYPE = 'PDS' for PDSs.
- &DSNTYPE is not specified. This indicates that the allocation request is provided by the user through JCL, the TSO/E ALLOCATE command, or dynamic allocation.

If you specify a DSNTYPE value in the JCL, and a different DSNTYPE value is also specified in the data class selected by ACS routines for the allocation, the value specified in the data class is ignored.

5.40 Allocating new PDSEs



```
//ALLOC   EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
    ALLOCATE
DSNAME('FILE.PDSE') -
          NEW  -
          DSNTYPE(LIBRARY)
```

Figure 5-40 Allocating a PDSE data set

Allocating new PDSEs

You can allocate PDSEs only in an SMS-managed environment. The PDSE data set does not have to be system-managed. To create a PDSE, use:

- DSNTYPE keyword in the JCL, TSO or IDCAMS **ALLOCATE** command
- A data class with LIBRARY in the Data Set Name Type field

You use DSNTYPE(LIBRARY) to allocate a PDSE, or DSNTYPE(PDS) to allocate a PDS. Figure 5-40 shows IDCAMS **ALLOCATE** used with the DSNTYPE(LIBRARY) keyword to allocate a PDSE.

A PDS and a PDSE can be concatenated in JCL DD statements, or by using dynamic allocation, such as the TSO **ALLOCATE** command.

5.41 System-managed data types

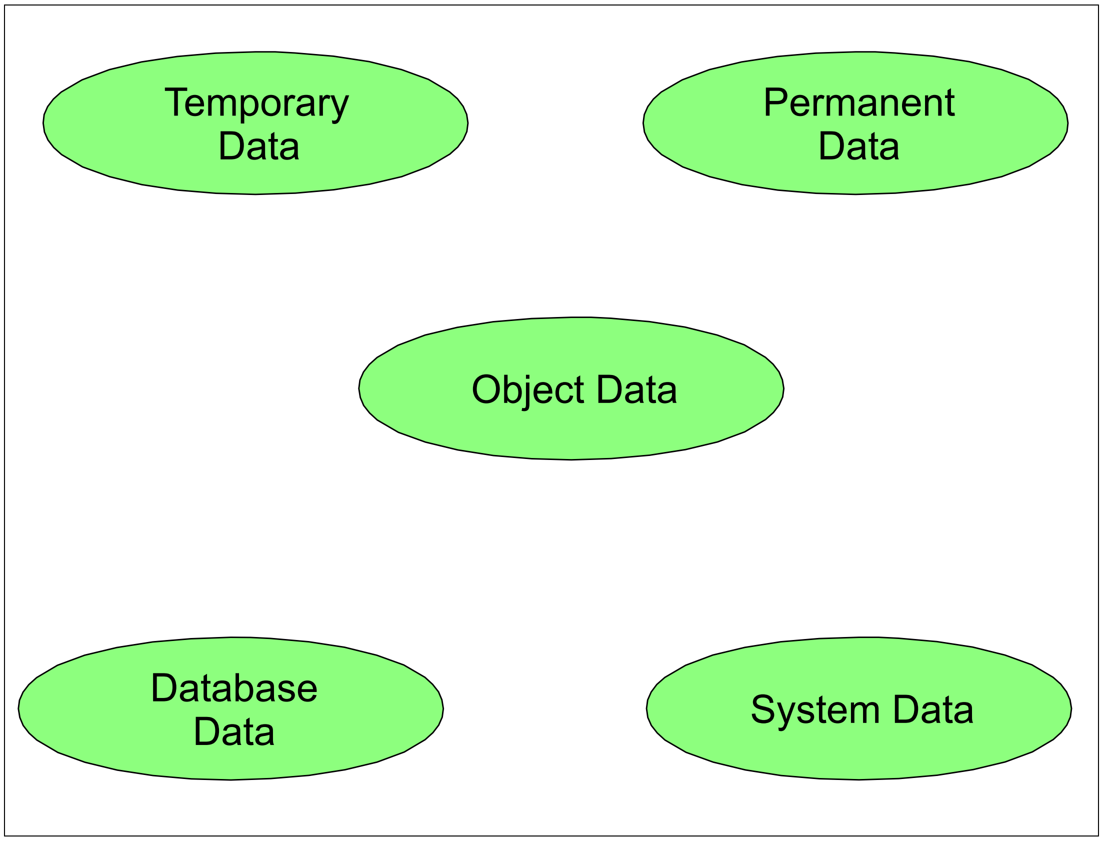


Figure 5-41 System-managed data types

Data set types that can be system-managed

Now that you have experience with SMS using the minimal SMS configuration, you can plan system-managed data sets implementation. First you need to know which data sets can be SMS-managed and which data sets cannot be SMS-managed.

These are some common types of data that can be system-managed. For details on how these data types can be system-managed using SMS storage groups, see *z/OS DFSMS Implementing System-Managed Storage,* SC26-7407.

Temporary data Data sets used only for the duration of a job, job step, or terminal session, and then deleted. These data sets can be cataloged or uncataloged, and can range in size from small to very large.

Permanent data Data sets consisting of:

- Interactive data
- TSO user data sets
- ISPF/PDF libraries you use during a terminal session

Data sets classified in this category are typically small, and are frequently accessed and updated.

Batch data Data that is classified as either online-initiated, production, or test.

- Data accessed as online-initiated are background jobs that an online facility (such as TSO) generates.

- Production batch refers to data created by specialized applications (such as payroll), that could be critical to the continued operation of your business or enterprise.
- Test batch refers to data created for testing purposes.

VSAM data	Data organized with VSAM, including VSAM data sets that are part of an existing database.
Large data	For most installations, large data sets occupy more than 10 percent of a single DASD volume. Note, however, that what constitutes a large data set is installation-dependent.
Multivolume data	Data sets that span more than one volume.
Database data	Data types usually having varied requirements for performance, availability, space, and security. To accommodate special needs, database products have specialized utilities to manage backup, recovery, and space usage. Examples include DB2, IMS, and CICS data.
System data	Data used by MVS to keep the operating system running smoothly. In a typical installation, 30 to 50 percent of these data sets are high performance and are used for cataloging, error recording, and other system functions. Because these critical data sets contain information required to find and access other data, they are read and updated frequently, often by more than one system in an installation. Performance and availability requirements are unique for system data. The performance of the system depends heavily upon the speed at which system data sets can be accessed. If a system data set such as a master catalog is unavailable, the availability of data across the entire system and across other systems can be affected. Some system data sets can be system-managed if they are uniquely named. These data sets include user catalogs. Place other system data sets on non-system managed volumes. The system data sets which are allocated at MVS system initialization are not system-managed, because the SMS address space is not active at initialization time.
Object data	Also known as byte-stream data, this data is used in specialized applications such as image processing, scanned correspondence, and seismic measurements. Object data typically has no internal record or field structure and, once written, the data is not changed or updated. However, the data can be referenced many times during its lifetime.

5.42 Data types that cannot be system-managed

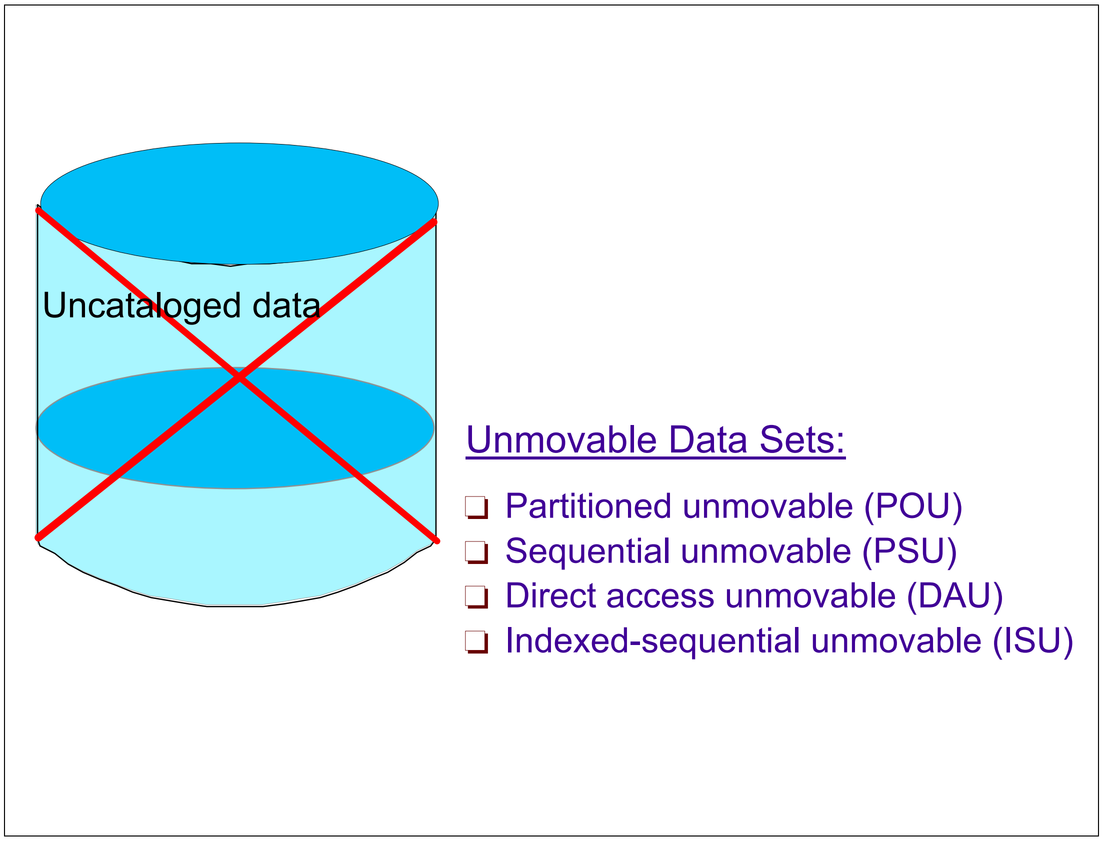

Uncataloged data
Unmovable Data Sets:
❑ Partitioned unmovable (POU)
❑ Sequential unmovable (PSU)
❑ Direct access unmovable (DAU)
❑ Indexed-sequential unmovable (ISU)

Figure 5-42 Data types that cannot be system-managed

Data types that cannot be system-managed

All permanent DASD data under the control of SMS *must* be cataloged in integrated catalog facility (ICF) catalogs using the standard search order. The catalogs contain the information required for locating and managing system-managed data sets.

When data sets are cataloged, users do not need to know which volumes the data sets reside on when they reference them; they do not need to specify unit type or volume serial number. This is essential in an environment with storage groups, where users do not have private volumes.

Some data cannot be system-managed, as described here:

- Uncataloged data

  Objects, stored in groups called collections, must have their collections cataloged in ICF catalogs because they, and the objects they contain, are system-managed data. The object access method (OAM) identifies an object by its collection name and the object's own name.

  An object is described only by an entry in a DB2 object directory. An object collection is described by a collection name catalog entry and a corresponding OAM collection identifier table entry. Therefore, an object is accessed by using the object's collection name and the catalog entry.

  When objects are written to tape, they are treated as tape data sets and OAM assigns two tape data set names to the objects. Objects in an object storage group being written to tape are stored as a tape data set named OAM.PRIMARY.DATA. Objects in an object backup storage group being written to tape are stored as a tape data set named OAM.BACKUP.DATA. Each tape containing objects has only one tape data set, and that data set has one of the two previous names. Because the same data set name can be used on multiple object-containing tape volumes, the object tape data sets are *not* cataloged.

If you do not already have a policy for cataloging all permanent data, it is a good idea to establish one now. For example, you can enforce standards by deleting uncataloged data sets.

- Uncataloged data sets

  Data sets that are not cataloged in any ICF catalog. To locate such a data set, you need to know the volume serial number of the volume on which the data set resides. SMS information is stored in the catalog, that’s why uncataloged data sets are not supported.

- Data sets in a jobcat or stepcat

  Data set LOCATEs using JOBCATs or STEPCATs are not permitted for system-managed data sets. You must identify the owning catalogs before you migrate these data sets to system management. The ISPF/PDF SUPERC utility is valuable for scanning your JCL and identifying any dependencies on JOBCATs or STEPCATs.

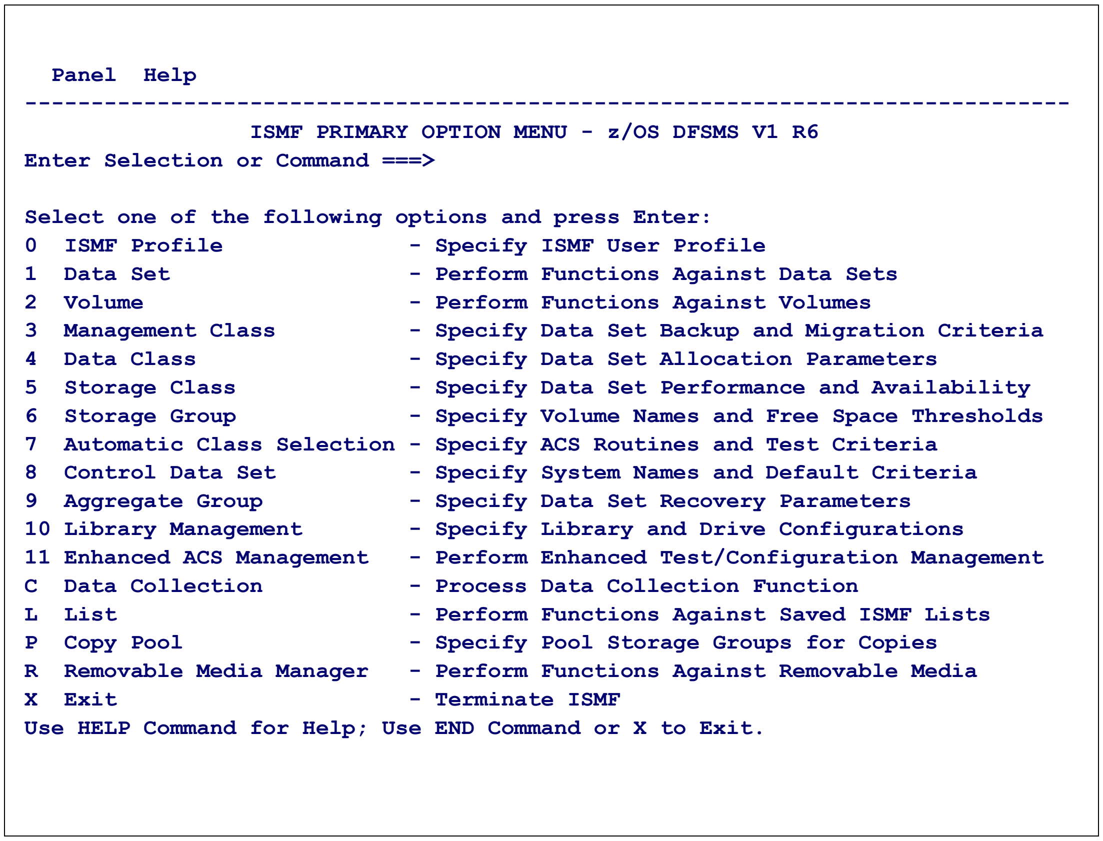
- Unmovable data sets

  Unmovable data sets cannot be system-managed. These data sets include:

  - Data sets identified by the following data set organizations (DSORGs):
    - Partitioned unmovable (POU)
    - Sequential unmovable (PSU)
    - Direct access unmovable (DAU)
    - Indexed-sequential unmovable (ISU)
  - Data sets with user-written access methods
  - Data sets containing processing control information about the device or volume on which they reside, including:
    - Absolute track data that is allocated in absolute DASD tracks or on split cylinders
    - Location-dependent direct data sets

  All unmovable data sets must be identified and converted for use in a system-managed environment. For information about identifying and converting unmovable data sets, see *z/OS DFSMSdss Storage Administration Guide*, SC35-0423.

5.43 Interactive Storage Management Facility (ISMF)



```
  Panel  Help
------------------------------------------------------------------------------
                 ISMF PRIMARY OPTION MENU - z/OS DFSMS V1 R6
Enter Selection or Command ===>

Select one of the following options and press Enter:
0  ISMF Profile              - Specify ISMF User Profile
1  Data Set                  - Perform Functions Against Data Sets
2  Volume                    - Perform Functions Against Volumes
3  Management Class          - Specify Data Set Backup and Migration Criteria
4  Data Class                - Specify Data Set Allocation Parameters
5  Storage Class             - Specify Data Set Performance and Availability
6  Storage Group             - Specify Volume Names and Free Space Thresholds
7  Automatic Class Selection - Specify ACS Routines and Test Criteria
8  Control Data Set          - Specify System Names and Default Criteria
9  Aggregate Group           - Specify Data Set Recovery Parameters
10 Library Management        - Specify Library and Drive Configurations
11 Enhanced ACS Management   - Perform Enhanced Test/Configuration Management
C  Data Collection           - Process Data Collection Function
L  List                      - Perform Functions Against Saved ISMF Lists
P  Copy Pool                 - Specify Pool Storage Groups for Copies
R  Removable Media Manager   - Perform Functions Against Removable Media
X  Exit                      - Terminate ISMF
Use HELP Command for Help; Use END Command or X to Exit.
```

Figure 5-43 ISMF Primary Option Menu panel

Interactive Storage Management Facility

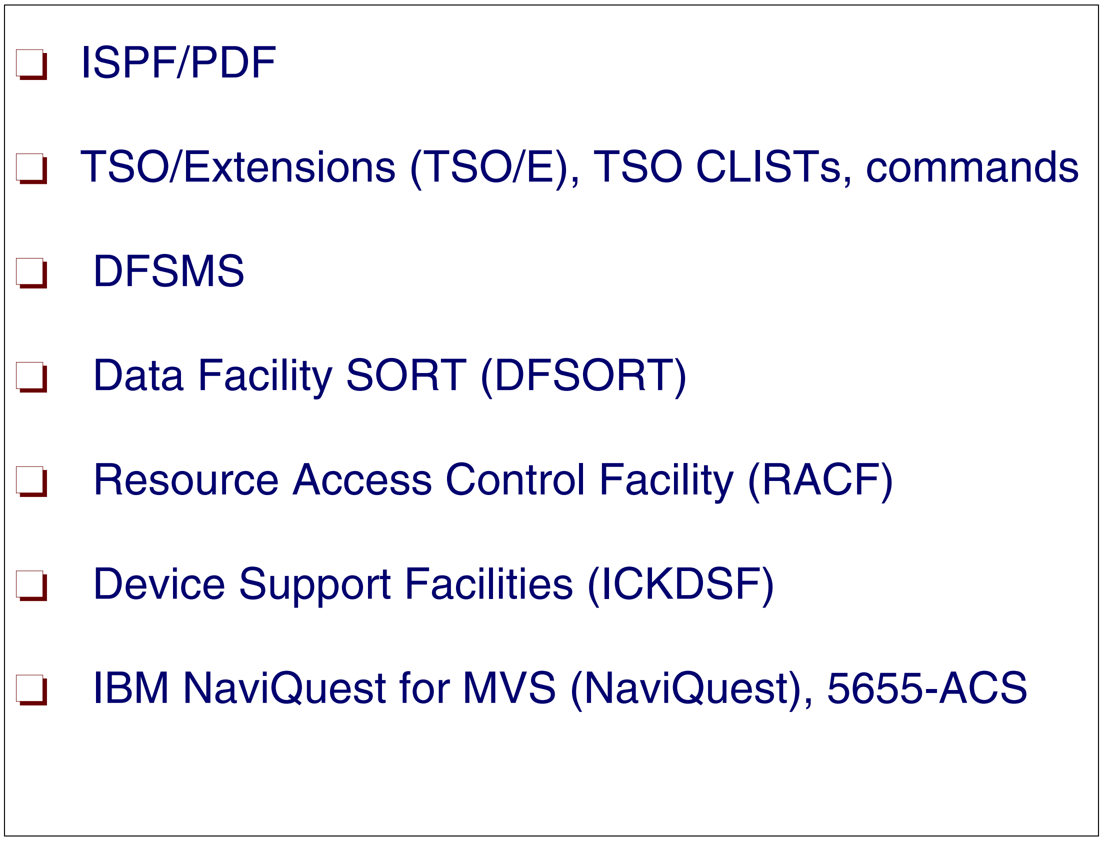
The Interactive Storage Management Facility (ISMF) helps you analyze and manage data and storage interactively. ISMF is an Interactive System Productivity Facility (ISPF) application. Figure 5-43 shows the first ISMF panel, the Primary Option Menu.

ISMF provides interactive access to the space management, backup, and recovery services of the DFSMShsm and DFSMSdss functional components of DFSMS, to the tape management services of the DFSMSrmm functional component, as well as to other products. DFSMS introduces the ability to use ISMF to define attributes of tape storage groups and libraries.

A storage administrator uses ISMF to define the installation's policy for managing storage by defining and managing SMS classes, groups, and ACS routines. ISMF then places the configuration in an SCDS. You can activate an SCDS through ISMF or an operator command.

ISMF is menu-driven, with fast paths for many of its functions. ISMF uses the ISPF data-tag language (DTL) to give its functional panels on workstations the look of common user access (CUA®) panels and a graphical user interface (GUI).

5.44 ISMF: Product relationships



- ISPF/PDF
- TSO/Extensions (TSO/E), TSO CLISTs, commands
- DFSMS
- Data Facility SORT (DFSORT)
- Resource Access Control Facility (RACF)
- Device Support Facilities (ICKDSF)
- IBM NaviQuest for MVS (NaviQuest), 5655-ACS

Figure 5-44 ISMF product relationships

ISMF product relationships

ISMF works with the following products, which you should be familiar with:

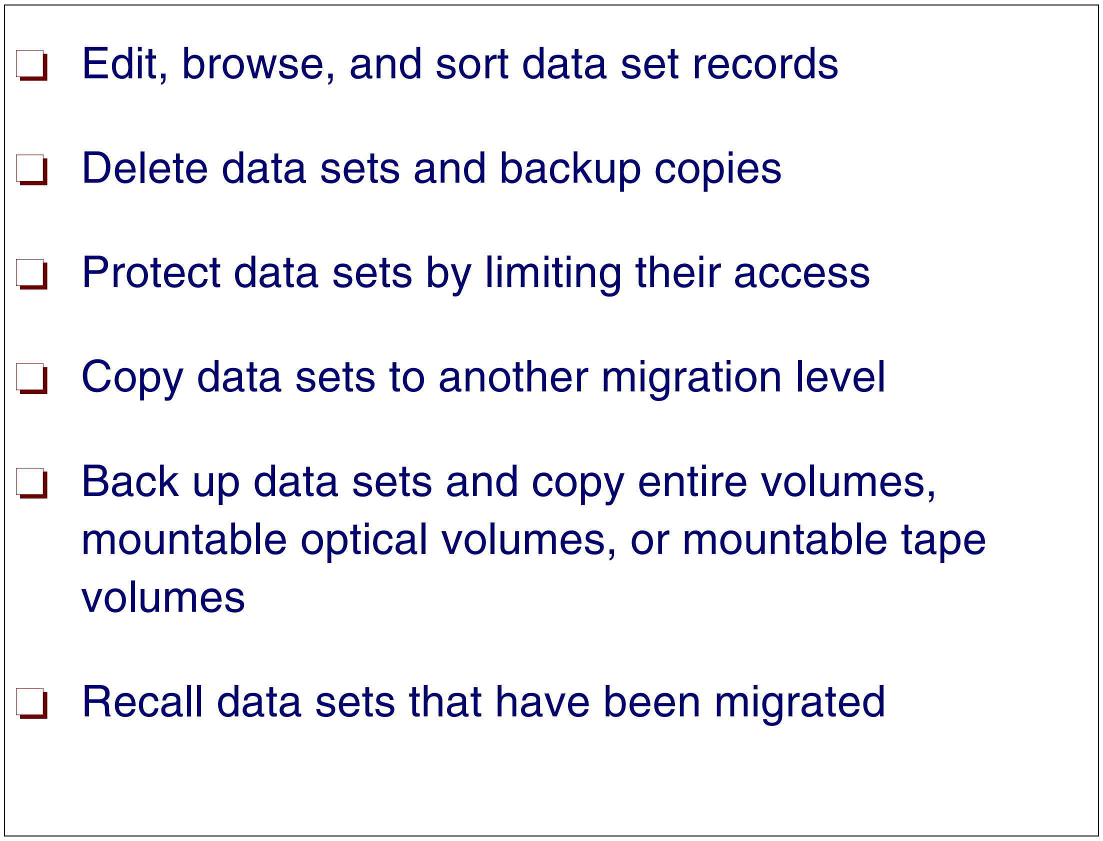
- Interactive System Productivity Facility/Program Development Facility (ISPF/PDF), which provides the edit, browse, data, and library utility functions.
- TSO/Extensions (TSO/E), TSO CLISTs and commands.
- DFSMS, which consists of five functional components: DFSMSdfp, DFSMShsm, DFSMSdss, DFSMSrmm and DFSMStvs. ISMF is designed to use the space management and availability management (backup/recovery) functions provided by those products.
- Data Facility SORT (DFSORT), which provides the record-level functions.
- Resource Access Control Facility (RACF), which provides the access control function for data and services.
- Device Support Facilities (ICKDSF) to provide the storage device support and analysis functions.
- IBM NaviQuest for MVS 5655-ACS.

  NaviQuest is a testing and reporting tool that speeds and simplifies the tasks associated with DFSMS initial implementation and ongoing ACS routine and configuration maintenance. NaviQuest assists storage administrators by allowing more automation of storage management tasks. More information about NaviQuest can be found in the NaviQuest User's Guide.

NaviQuest provides:

- A familiar ISPF panel interface to functions
- Fast, easy, bulk test-case creation
- ACS routine and DFSMS configuration-testing automation
- Storage reporting assistance
- Additional tools to aid with storage administration tasks
- Batch creation of data set and volume listings
- Printing of ISMF LISTs
- Batch ACS routine translation
- Batch ACS routine validation

5.45 ISMF: What you can do with ISMF



- Edit, browse, and sort data set records
- Delete data sets and backup copies
- Protect data sets by limiting their access
- Copy data sets to another migration level
- Back up data sets and copy entire volumes, mountable optical volumes, or mountable tape volumes
- Recall data sets that have been migrated

Figure 5-45 What you can do with ISMF

What you can do with ISMF

ISMF is a panel-driven interface. Use the panels in an ISMF application to:

- Display lists with information about specific data sets, DASD volumes, mountable optical volumes, and mountable tape volumes
- Generate lists of data, storage, and management classes to find out how data sets are being managed
- Display and manage lists saved from various ISMF applications

ISMF generates a data list based on your selection criteria. Once the list is built, you can use ISMF entry panels to perform space management or backup and recovery tasks against the entries in the list.

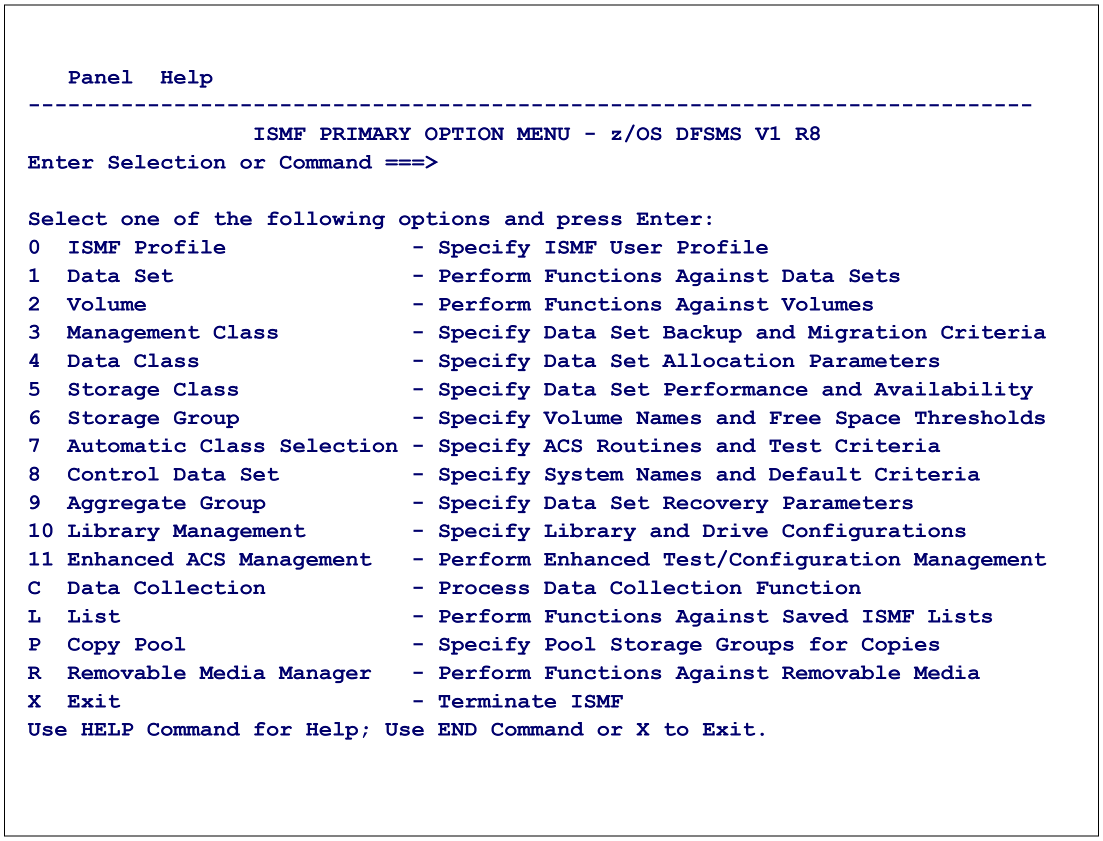
As a user performing data management tasks against individual data sets or against lists of data sets or volumes, you can use ISMF to:

- Edit, browse, and sort data set records
- Delete data sets and backup copies
- Protect data sets by limiting their access
- Recover unused space from data sets and consolidate free space on DASD volumes
- Copy data sets or DASD volumes to the same device or another device
- Migrate data sets to another migration level

- Recall data sets that have been migrated so that they can be used
- Back up data sets and copy entire volumes for availability purposes
- Recover data sets and restore DASD volumes, mountable optical volumes, or mountable tape volumes

You cannot allocate data sets from ISMF. Data sets are allocated from ISPF, from TSO, or with JCL statements. ISMF provides the **DSUTIL** command, which enables users to get to ISPF and toggle back to ISMF.

5.46 ISMF: Accessing ISMF



```
   Panel  Help
----------------------------------------------------------------------------
                 ISMF PRIMARY OPTION MENU - z/OS DFSMS V1 R8
Enter Selection or Command ===>

Select one of the following options and press Enter:
0  ISMF Profile             - Specify ISMF User Profile
1  Data Set                 - Perform Functions Against Data Sets
2  Volume                   - Perform Functions Against Volumes
3  Management Class         - Specify Data Set Backup and Migration Criteria
4  Data Class               - Specify Data Set Allocation Parameters
5  Storage Class            - Specify Data Set Performance and Availability
6  Storage Group            - Specify Volume Names and Free Space Thresholds
7  Automatic Class Selection - Specify ACS Routines and Test Criteria
8  Control Data Set         - Specify System Names and Default Criteria
9  Aggregate Group          - Specify Data Set Recovery Parameters
10 Library Management       - Specify Library and Drive Configurations
11 Enhanced ACS Management  - Perform Enhanced Test/Configuration Management
C  Data Collection          - Process Data Collection Function
L  List                     - Perform Functions Against Saved ISMF Lists
P  Copy Pool                - Specify Pool Storage Groups for Copies
R  Removable Media Manager  - Perform Functions Against Removable Media
X  Exit                     - Terminate ISMF
Use HELP Command for Help; Use END Command or X to Exit.
```

Figure 5-46 ISMF Primary Option Menu panel for storage administrator mode

Accessing ISMF

How you access ISMF depends on your site.

- You can create an option on the ISPF Primary Option Menu to access ISMF. Then access ISMF by typing the appropriate option after the arrow on the Option field, in the ISPF Primary Option Menu. This starts an ISMF session from the ISPF/PDF Primary Option Menu.
- To access ISMF directly from TSO, use the command:

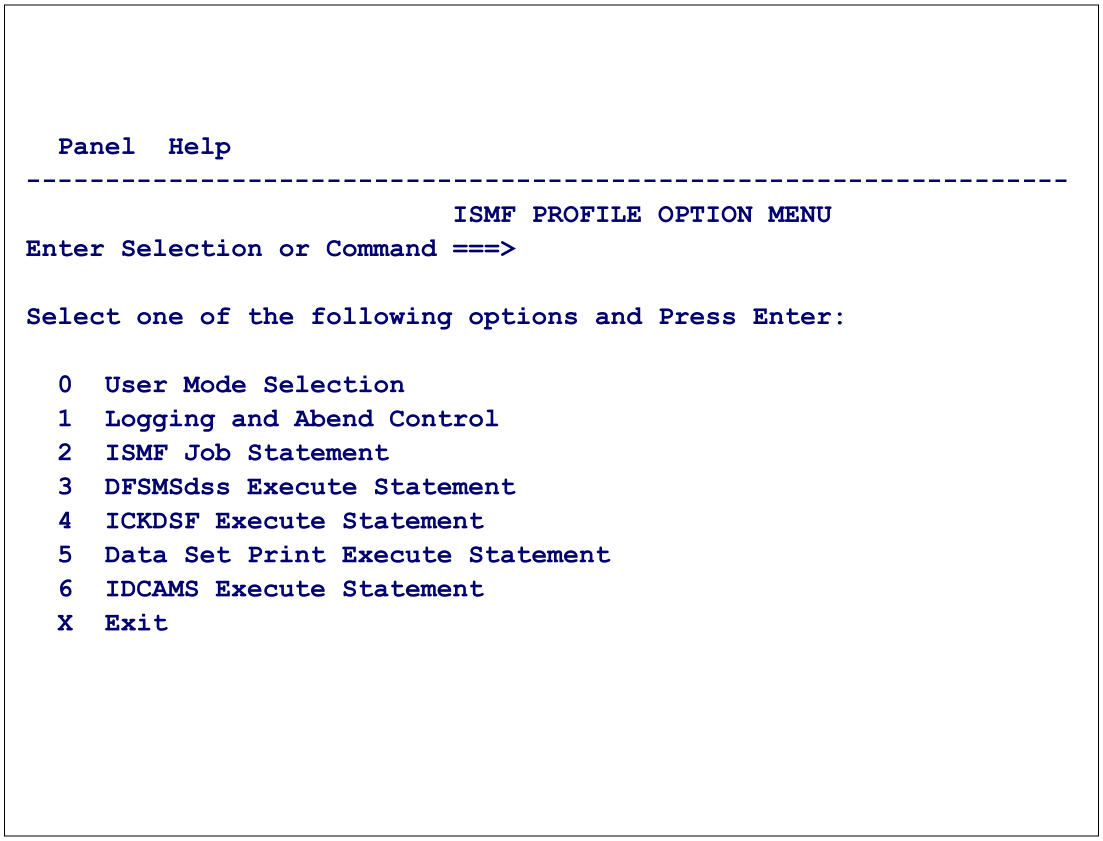
  ISPSTART PGM(DGTFMD01) NEWAPPL(DGT)

There are two Primary Option Menus, one for storage administrators, and another for end users. Figure 5-46 shows the menu available to storage administrators; it includes additional applications not available to end users.

Option 0 controls the user mode or the type of Primary Option Menu to be displayed. Refer to “ISMF: Profile option” on page 295 for information about how to change the user mode.

The ISMF Primary Option Menu example assumes installation of DFSMS at the current release level. For information about adding the DFSORT option to your Primary Option Menu, refer to *DFSORT Installation and Customization Release 14,* SC33-4034.

5.47 ISMF: Profile option



```
  Panel  Help
------------------------------------------------------------------
                           ISMF PROFILE OPTION MENU
Enter Selection or Command ===>

Select one of the following options and Press Enter:

  0  User Mode Selection
  1  Logging and Abend Control
  2  ISMF Job Statement
  3  DFSMSdss Execute Statement
  4  ICKDSF Execute Statement
  5  Data Set Print Execute Statement
  6  IDCAMS Execute Statement
  X  Exit
```

Figure 5-47 ISMF PROFILE OPTION MENU panel

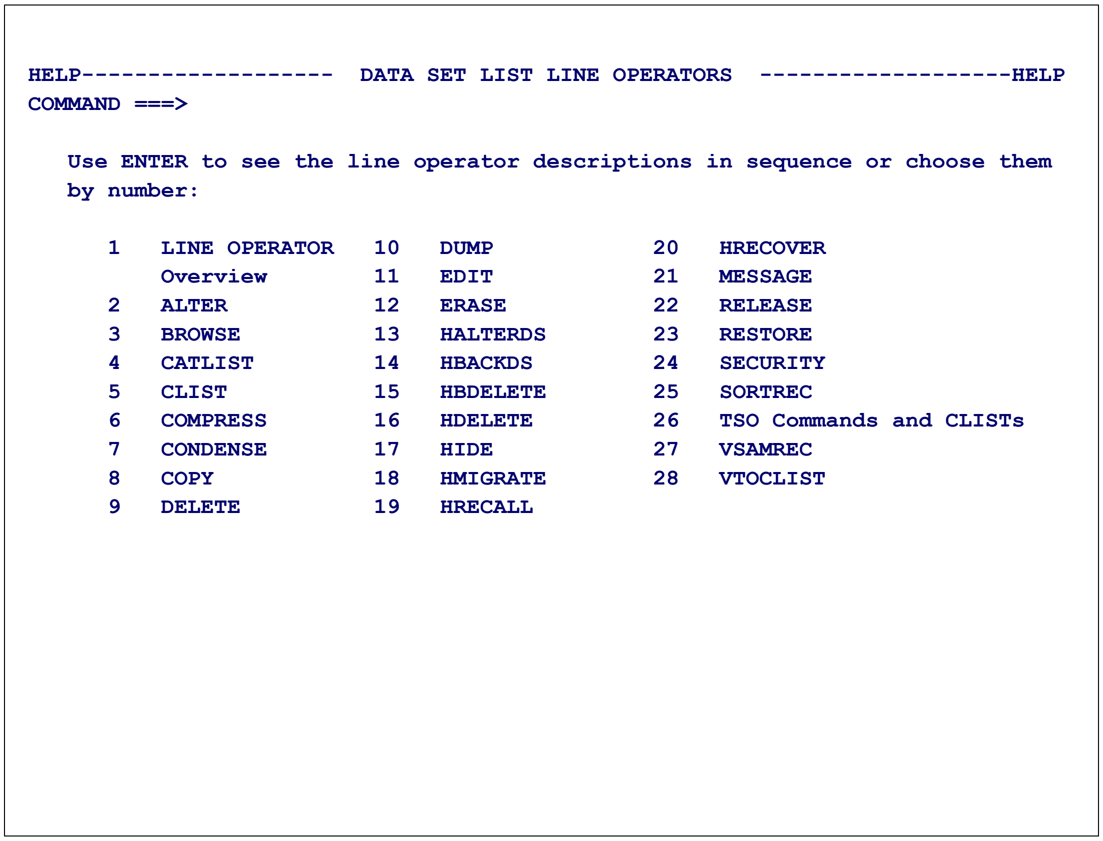
Setting the ISMF profile

Figure 5-47 shows the ISMF Profile Option Menu panel, option **0** from the ISMF Primary Menu. Use this menu to control the way ISMF runs during the session. You can:

- Change the user mode from end user to storage administrator, or from storage administrator to end user
- Control ISMF error logging and recovery from abends
- Define statements for ISMF to use in processing your jobs, such as:
  - JOB statements
  - DFSMSdss
  - Device Support Facilities (ICKDSF)
  - Access Method Services (IDCAMS)
  - PRINT execute statements in your profile

You can select ISMF or ISPF JCL statements for processing batch jobs.

5.48 ISMF: Obtaining information about a panel field



```
HELP-------------------  DATA SET LIST LINE OPERATORS  -------------------HELP
COMMAND ===>

   Use ENTER to see the line operator descriptions in sequence or choose them
   by number:

      1   LINE OPERATOR   10   DUMP           20   HRECOVER
          Overview        11   EDIT           21   MESSAGE
      2   ALTER           12   ERASE          22   RELEASE
      3   BROWSE          13   HALTERDS       23   RESTORE
      4   CATLIST         14   HBACKDS        24   SECURITY
      5   CLIST           15   HBDELETE       25   SORTREC
      6   COMPRESS        16   HDELETE        26   TSO Commands and CLISTs
      7   CONDENSE        17   HIDE           27   VSAMREC
      8   COPY            18   HMIGRATE       28   VTOCLIST
      9   DELETE          19   HRECALL
```

Figure 5-48 Obtaining information using Help command

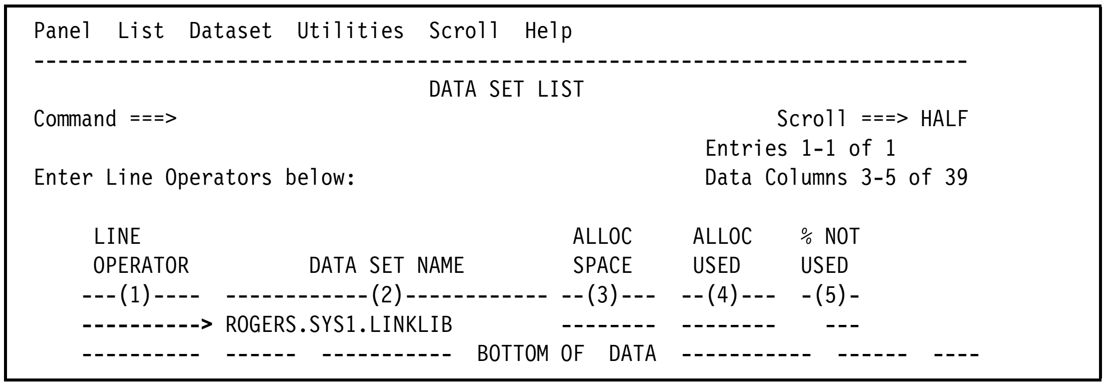
Using the Help program function key

On any ISMF panel, you can use the ISMF Help program function key (PF key) to obtain information about the panel you are using and the panel fields. By positioning the cursor in a specific field, you can obtain detailed information related to that field.

Figure 5-48 shows the panel you reach when you press the Help PF key with the cursor in the Line Operator field of the panel shown in Figure 5-49 on page 297 where the arrow points to the data set. The Data Set List Line Operators panel shows the commands available to enter in that field. If you want an explanation about a specific command, type the option corresponding to the desired command and a panel is displayed showing information about the command function.

You can exploit the Help PF key, when defining classes, to obtain information about what you have to enter in the fields. Place the cursor in the field and press the Help PF key.

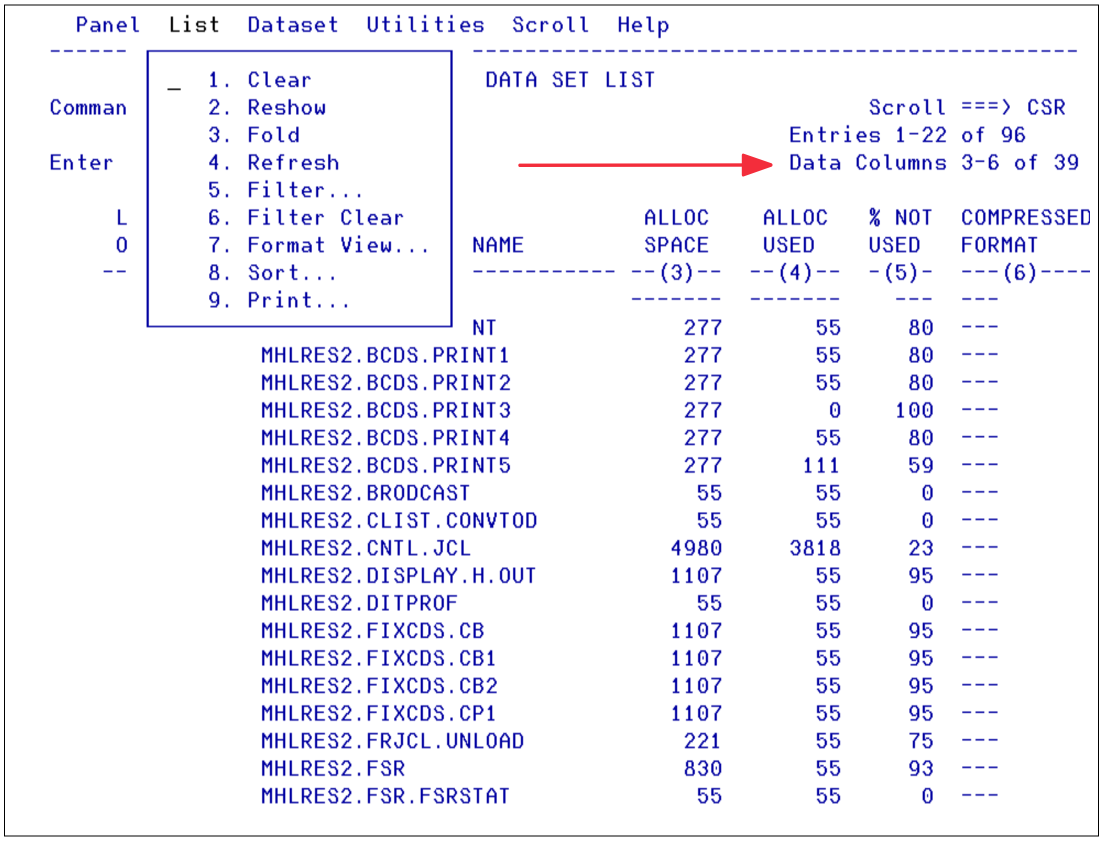
To see and change the assigned functions to the PF keys, enter the **KEYS** command in the Command field.



```
Panel  List  Dataset  Utilities  Scroll  Help
------------------------------------------------------------------------------
                                 DATA SET LIST
Command ===>                                                Scroll ===> HALF
                                                          Entries 1-1 of 1
Enter Line Operators below:                               Data Columns 3-5 of 39

     LINE                                   ALLOC     ALLOC    % NOT
     OPERATOR         DATA SET NAME         SPACE     USED     USED
    ---(1)----  ------------(2)------------ --(3)---  --(4)---  -(5)-
    ----------> ROGERS.SYS1.LINKLIB         --------  --------    ---
    ----------  ------  -----------  BOTTOM OF  DATA  -----------  ------  ----
```

Figure 5-49 Data Set List panel

5.49 ISMF: Data set option



```
  Panel  List  Dataset  Utilities  Scroll  Help
------ ┌──────────────────────┐ ---------------------------------------------
       │ _   1. Clear         │  DATA SET LIST
Comman │     2. Reshow        │                              Scroll ===> CSR
       │     3. Fold          │                        Entries 1-22 of 96
Enter  │     4. Refresh       │                        Data Columns 3-6 of 39
       │     5. Filter...     │
     L │     6. Filter Clear  │               ALLOC    ALLOC   % NOT  COMPRESSED
     O │     7. Format View...│ NAME          SPACE    USED    USED   FORMAT
    -- │     8. Sort...       │ ----------- --(3)-- --(4)-- -(5)-  ---(6)----
       │     9. Print...      │             -------  -------  ---   ---
       └──────────────────────┘ NT              277       55     80  ---
                MHLRES2.BCDS.PRINT1             277       55     80  ---
                MHLRES2.BCDS.PRINT2             277       55     80  ---
                MHLRES2.BCDS.PRINT3             277        0    100  ---
                MHLRES2.BCDS.PRINT4             277       55     80  ---
                MHLRES2.BCDS.PRINT5             277      111     59  ---
                MHLRES2.BRODCAST                 55       55      0  ---
                MHLRES2.CLIST.CONVTOD            55       55      0  ---
                MHLRES2.CNTL.JCL               4980     3818     23  ---
                MHLRES2.DISPLAY.H.OUT          1107       55     95  ---
                MHLRES2.DITPROF                  55       55      0  ---
                MHLRES2.FIXCDS.CB              1107       55     95  ---
                MHLRES2.FIXCDS.CB1             1107       55     95  ---
                MHLRES2.FIXCDS.CB2             1107       55     95  ---
                MHLRES2.FIXCDS.CP1             1107       55     95  ---
                MHLRES2.FRJCL.UNLOAD            221       55     75  ---
                MHLRES2.FSR                     830       55     93  ---
                MHLRES2.FSR.FSRSTAT              55       55      0  ---
```

Figure 5-50 Data Set List panel using action bars

Data Set Selection Entry panel

When you select option **1** (Data Set) from the ISMF Primary Option Menu, you get to the Data Set Selection Entry Panel. There you can specify filter criteria to get a list of data sets, as follows:

```
2  Generate a new list from criteria below
     Data Set Name . . . 'MHLRES2.**'
```

Figure 5-50 shows the data set list generated for the generic data set name MHLRES2.**.

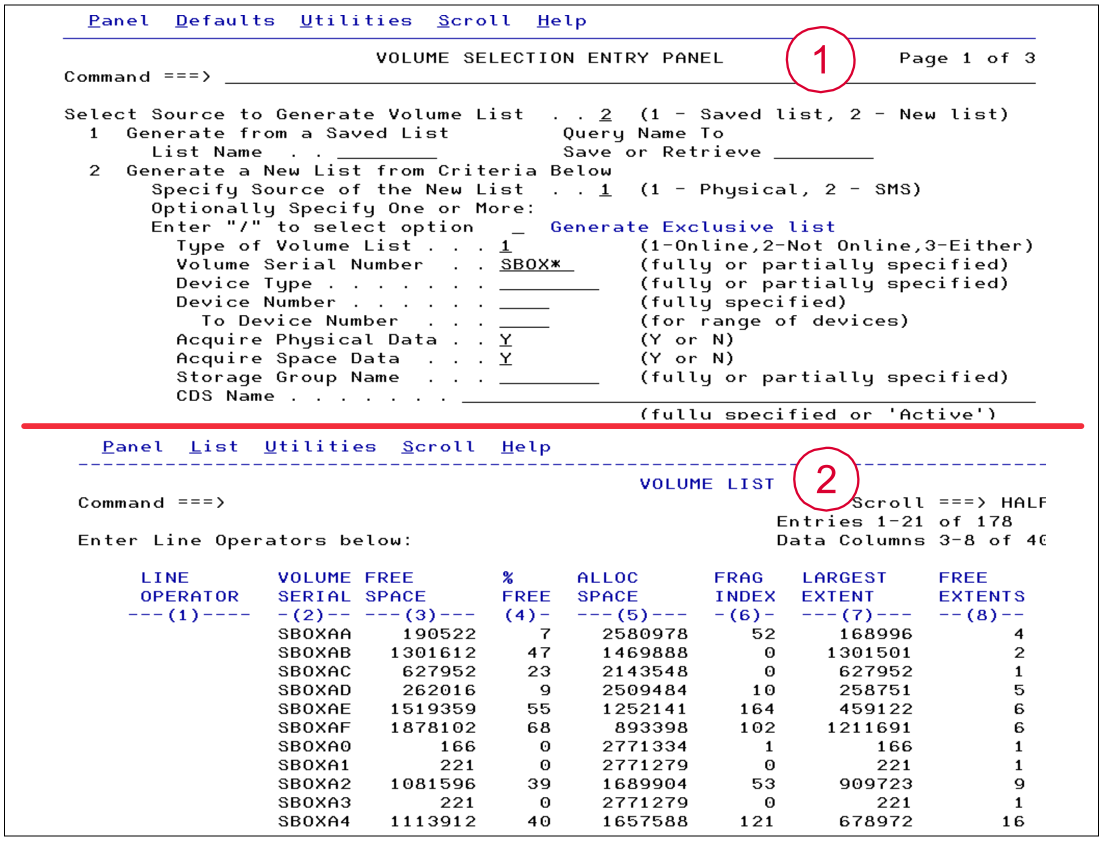
Data Set List panel

You can use line operators to execute tasks with individual data sets. Use list commands to execute tasks with a group of data sets. These tasks include editing, browsing, recovering unused space, copying, migrating, deleting, backing up, and restoring data sets. TSO commands and CLISTs can also be used as line operators or list commands. You can save a copy of a data set list and reuse it later.

If ISMF is unable to get certain information required to check if a data set meets the selection criteria specified, that data set is also to be included in the list. Missing information is indicated by dashes on the corresponding column.

The Data Fields field shows how many fields you have in the list. You can navigate throughout these fields using Right and Left PF keys. The figure also shows the use of the actions bar.

5.50 ISMF: Volume Option



```
  Panel  Defaults  Utilities  Scroll  Help
------------------------------------------------------------------------------
                         VOLUME SELECTION ENTRY PANEL              Page 1 of 3
Command ===> _________________________________________________________________

Select Source to Generate Volume List  . . 2  (1 - Saved list, 2 - New list)
  1  Generate from a Saved List          Query Name To
       List Name  . . ________           Save or Retrieve ________
  2  Generate a New List from Criteria Below
       Specify Source of the New List  . . 1  (1 - Physical, 2 - SMS)
       Optionally Specify One or More:
       Enter "/" to select option    _  Generate Exclusive list
         Type of Volume List . . . 1          (1-Online,2-Not Online,3-Either)
         Volume Serial Number  . . SBOX*      (fully or partially specified)
         Device Type . . . . . . . ________   (fully or partially specified)
         Device Number . . . . . . ____       (fully specified)
           To Device Number  . . . ____       (for range of devices)
         Acquire Physical Data . . Y          (Y or N)
         Acquire Space Data  . . . Y          (Y or N)
         Storage Group Name  . . . ________   (fully or partially specified)
         CDS Name  . . . . . . . ______________________________________________
                                              (fully specified or 'Active')
```

```
  Panel  List  Utilities  Scroll  Help
------------------------------------------------------------------------------
                                  VOLUME LIST
Command ===>                                               Scroll ===> HALF
                                                      Entries 1-21 of 178
Enter Line Operators below:                           Data Columns 3-8 of 40

     LINE        VOLUME FREE       %    ALLOC      FRAG  LARGEST    FREE
     OPERATOR    SERIAL SPACE      FREE SPACE      INDEX EXTENT     EXTENTS
    ---(1)----   -(2)-- ---(3)---  (4)- ---(5)---  -(6)- ---(7)---  --(8)--
                 SBOXAA    190522    7   2580978    52    168996        4
                 SBOXAB   1301612   47   1469888     0   1301501        2
                 SBOXAC    627952   23   2143548     0    627952        1
                 SBOXAD    262016    9   2509484    10    258751        5
                 SBOXAE   1519359   55   1252141   164    459122        6
                 SBOXAF   1878102   68    893398   102   1211691        6
                 SBOXA0       166    0   2771334     1       166        1
                 SBOXA1       221    0   2771279     0       221        1
                 SBOXA2   1081596   39   1689904    53    909723        9
                 SBOXA3       221    0   2771279     0       221        1
                 SBOXA4   1113912   40   1657588   121    678972       16
```

Figure 5-51 Volume Selection Entry panel

Volume option

Selecting option **2** (Volume) from the ISMF Primary Option Menu takes you to the Volume List Selection Menu panel, as follows:

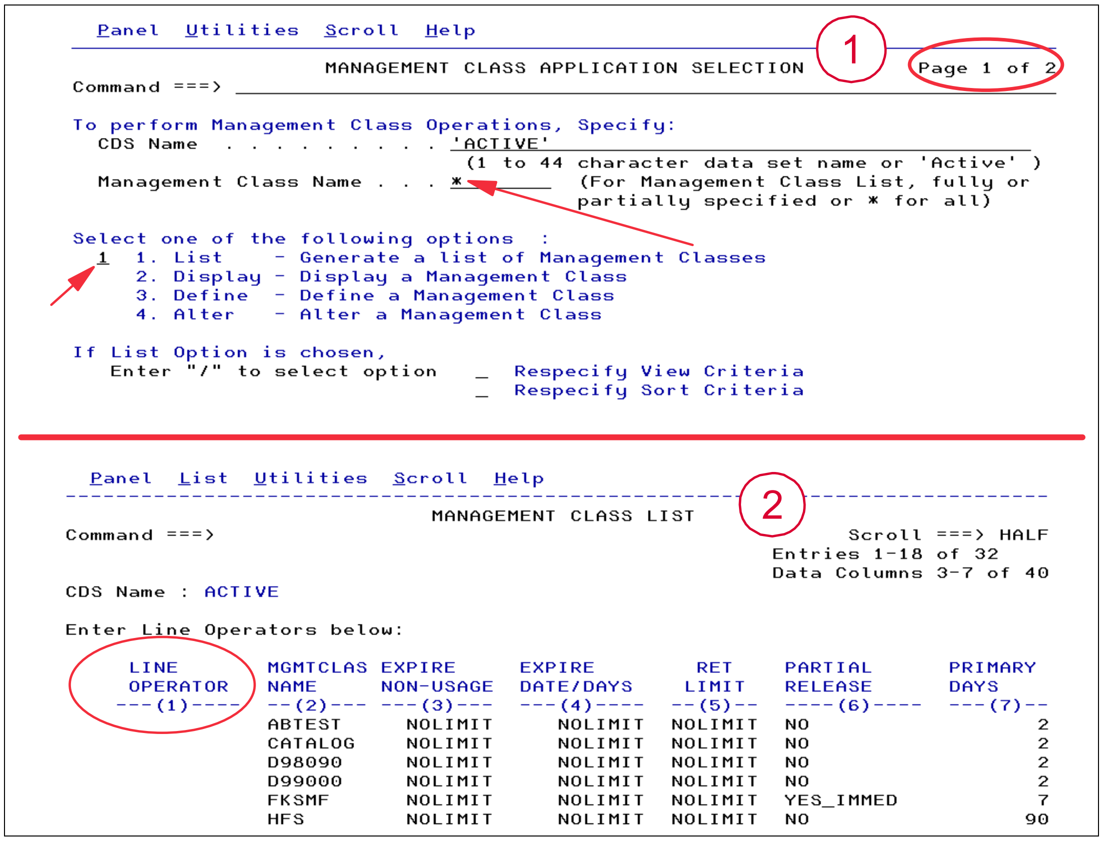
```
                         VOLUME LIST SELECTION MENU
Enter Selection or Command ===> _____________________________________________

1  DASD                     - Generate a List of DASD Volumes
2  Mountable Optical        - Generate a List of Mountable Optical Volume
3  Mountable Tape           - Generate a List of Mountable Tape Volumes
```

Selecting option **1** (DASD) displays the Volume Selection Entry Panel, shown in part (1) of Figure 5-51. Using filters, you can select a Volume List Panel, shown in part (2) of the figure.

Volume List panel

The volume application constructs a list of the type you choose in the Volume List Selection Menu. Use line operators to do tasks with an individual volume. These tasks include consolidating or recovering unused space, copying, backing up, and restoring volumes. TSO commands and CLISTs can also be line operators or list commands. You can save a copy of a volume list and reuse it later. With the list of mountable optical volumes or mountable tape volumes, you can only browse the list.

5.51 ISMF: Management Class option



```
  Panel  Utilities  Scroll  Help
 ------------------------------------------------------------------------------
                   MANAGEMENT CLASS APPLICATION SELECTION            Page 1 of 2
 Command ===> ____________________________________________________________

 To perform Management Class Operations, Specify:
   CDS Name  . . . . . . . . . 'ACTIVE'
                                (1 to 44 character data set name or 'Active' )
   Management Class Name . . . *         (For Management Class List, fully or
                                          partially specified or * for all)

 Select one of the following options  :
   1   1. List     - Generate a list of Management Classes
       2. Display  - Display a Management Class
       3. Define   - Define a Management Class
       4. Alter    - Alter a Management Class

 If List Option is chosen,
    Enter "/" to select option      _  Respecify View Criteria
                                    _  Respecify Sort Criteria
```

```
  Panel  List  Utilities  Scroll  Help
 ------------------------------------------------------------------------------
                              MANAGEMENT CLASS LIST
 Command ===>                                                Scroll ===> HALF
                                                       Entries 1-18 of 32
                                                       Data Columns 3-7 of 40
 CDS Name : ACTIVE

 Enter Line Operators below:

      LINE        MGMTCLAS EXPIRE      EXPIRE       RET     PARTIAL      PRIMARY
      OPERATOR    NAME     NON-USAGE   DATE/DAYS    LIMIT   RELEASE      DAYS
     ---(1)----   --(2)--- ---(3)---   ---(4)----   --(5)-- ----(6)----  ---(7)--
                  ABTEST     NOLIMIT      NOLIMIT   NOLIMIT NO                  2
                  CATALOG    NOLIMIT      NOLIMIT   NOLIMIT NO                  2
                  D98090     NOLIMIT      NOLIMIT   NOLIMIT NO                  2
                  D99000     NOLIMIT      NOLIMIT   NOLIMIT NO                  2
                  FKSMF      NOLIMIT      NOLIMIT   NOLIMIT YES_IMMED           7
                  HFS        NOLIMIT      NOLIMIT   NOLIMIT NO                 90
```

Figure 5-52 Management Class Selection Menu panel

Management Class Application Selection panel

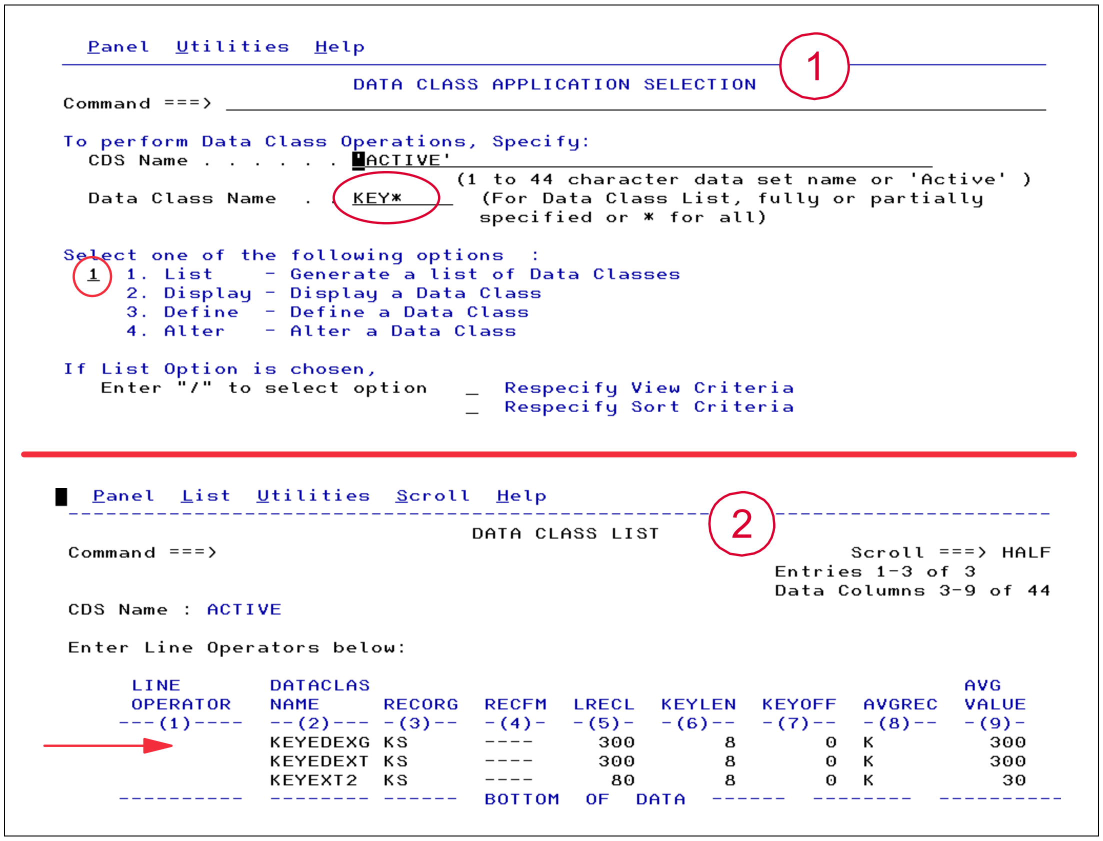
The first panel (1) in Figure 5-52 shows the panel displayed when you select option **3** (Management Class) from the ISMF Primary Option Menu. Use this option to display, modify, and define options for the SMS management classes. It also constructs a list of the available management classes.

Management Class List panel

The second panel (2) in Figure 5-52 shows the management class list generated by the filters chosen in the previous panel, using option **1** (List). Note how many data columns are available. You can navigate through them using the Right and Left PF keys.

To view the commands, you can use in the Line Operator field (marked with a circle in the figure), place the cursor in the field and press the Help PF key.

5.52 ISMF: Data Class option



```
  Panel  Utilities  Help
                       DATA CLASS APPLICATION SELECTION
Command ===>

To perform Data Class Operations, Specify:
  CDS Name . . . . . . 'ACTIVE'
                                 (1 to 44 character data set name or 'Active' )
  Data Class Name  . . KEY*        (For Data Class List, fully or partially
                                   specified or * for all)

Select one of the following options :
  1  1. List     - Generate a list of Data Classes
     2. Display  - Display a Data Class
     3. Define   - Define a Data Class
     4. Alter    - Alter a Data Class

If List Option is chosen,
   Enter "/" to select option      _  Respecify View Criteria
                                   _  Respecify Sort Criteria
```

```
   Panel  List  Utilities  Scroll  Help
 ------------------------------------------------------------------------------
                                DATA CLASS LIST
 Command ===>                                                  Scroll ===> HALF
                                                             Entries 1-3 of 3
                                                             Data Columns 3-9 of 44
 CDS Name : ACTIVE

 Enter Line Operators below:

      LINE       DATACLAS                                                  AVG
      OPERATOR   NAME      RECORG  RECFM  LRECL  KEYLEN  KEYOFF  AVGREC  VALUE
     ---(1)----  --(2)---  -(3)--  -(4)-  -(5)-  -(6)--  -(7)--  -(8)--  -(9)-
                 KEYEDEXG  KS      ----     300       8       0  K         300
                 KEYEDEXT  KS      ----     300       8       0  K         300
                 KEYEXT2   KS      ----      80       8       0  K          30
     ----------  --------  ------  BOTTOM  OF   DATA  ------  --------  ----------
```

Figure 5-53 Data Class Application Selection panel

Displaying information about data classes

The first panel (1) in Figure 5-53 is the panel displayed when you choose option **4** (Data Class) from the ISMF Primary Option Menu. Use this option to define the way data sets are allocated in your installation.

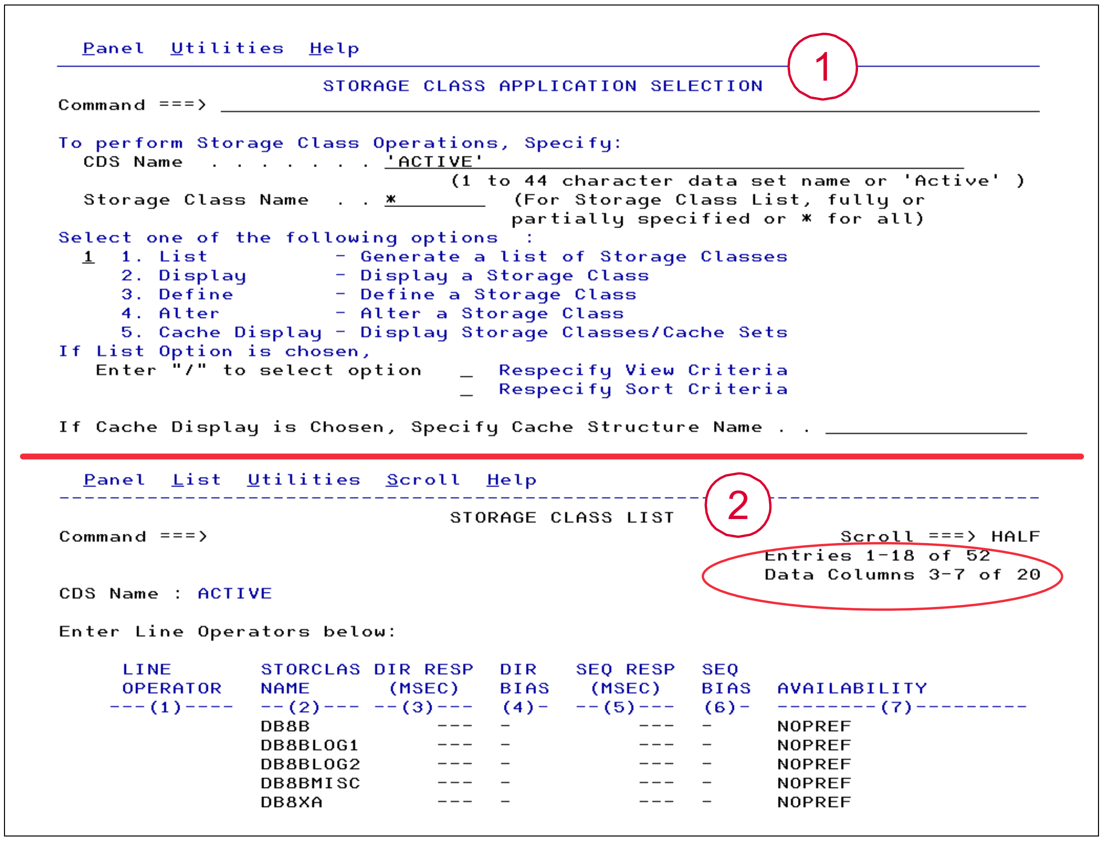
Data class attributes are assigned to a data set when the data set is created. They apply to *both* SMS-managed and non-SMS-managed data sets. Attributes specified in JCL or equivalent allocation statements override those specified in a data class. Individual attributes in a data class can be overridden by JCL, TSO, IDCAMS, and dynamic allocation statements.

Data Class List panel

The second panel (2) in Figure 5-53 is the Data Class List generated by the filters specified in the previous panel.

Entering the **DISPLAY** line command in the Line Operator field, in front of a data class name, displays the information about that data class, without requiring you to navigate using the Right and Left PF keys.

5.53 ISMF: Storage Class option


```
  Panel  Utilities  Help
                       STORAGE CLASS APPLICATION SELECTION            1
Command ===> ____________________________________________________________

To perform Storage Class Operations, Specify:
  CDS Name  . . . . . . . 'ACTIVE'_____________________________________
                               (1 to 44 character data set name or 'Active' )
  Storage Class Name  . . *_______    (For Storage Class List, fully or
                                      partially specified or * for all)
Select one of the following options  :
  1  1. List          - Generate a list of Storage Classes
     2. Display       - Display a Storage Class
     3. Define        - Define a Storage Class
     4. Alter         - Alter a Storage Class
     5. Cache Display - Display Storage Classes/Cache Sets
If List Option is chosen,
   Enter "/" to select option     _  Respecify View Criteria
                                  _  Respecify Sort Criteria

If Cache Display is Chosen, Specify Cache Structure Name . . ________________
```

```
  Panel  List  Utilities  Scroll  Help
----------------------------------------------------- 2 ---------------------
                               STORAGE CLASS LIST
Command ===>                                                 Scroll ===> HALF
                                                         Entries 1-18 of 52
                                                         Data Columns 3-7 of 20
CDS Name : ACTIVE

Enter Line Operators below:

     LINE       STORCLAS DIR RESP  DIR   SEQ RESP  SEQ
     OPERATOR   NAME      (MSEC)   BIAS   (MSEC)   BIAS  AVAILABILITY
    ---(1)----  --(2)--- --(3)---  (4)-  --(5)---  (6)-  --------(7)---------
                DB8B          ---  -          ---  -     NOPREF
                DB8BLOG1      ---  -          ---  -     NOPREF
                DB8BLOG2      ---  -          ---  -     NOPREF
                DB8BMISC      ---  -          ---  -     NOPREF
                DB8XA         ---  -          ---  -     NOPREF
```

Figure 5-54 Storage Class Application Selection panel

Storage Class Application Selection panel

The first panel (1) in Figure 5-54 shows the Storage Class Application Selection panel that is displayed when you select option **5** (Storage Class) of the ISMF Primary Option Menu.

The Storage Class Application Selection panel lets the storage administrator specify performance objectives and availability attributes that characterize a collection of data sets.

For objects, the storage administrator can define the performance attribute Initial Access Response Seconds. A data set or object must be assigned to a storage class in order to be managed by DFSMS.

Storage Class List panel

The second panel (2) in Figure 5-54 shows the storage class list generated by the filters specified in the previous panel.

You can specify the **`DISPLAY`** line operator next to any class name on a class list to generate a panel that displays values associated with that particular class. This information can help you decide whether you need to assign a new DFSMS class to your data set or object.

If you determine that a data set you own should be associated with a different management class or storage class, and if you have authorization, you can use the **`ALTER`** line operator against a data set list entry to specify another storage class or management class.

5.54 ISMF: List option

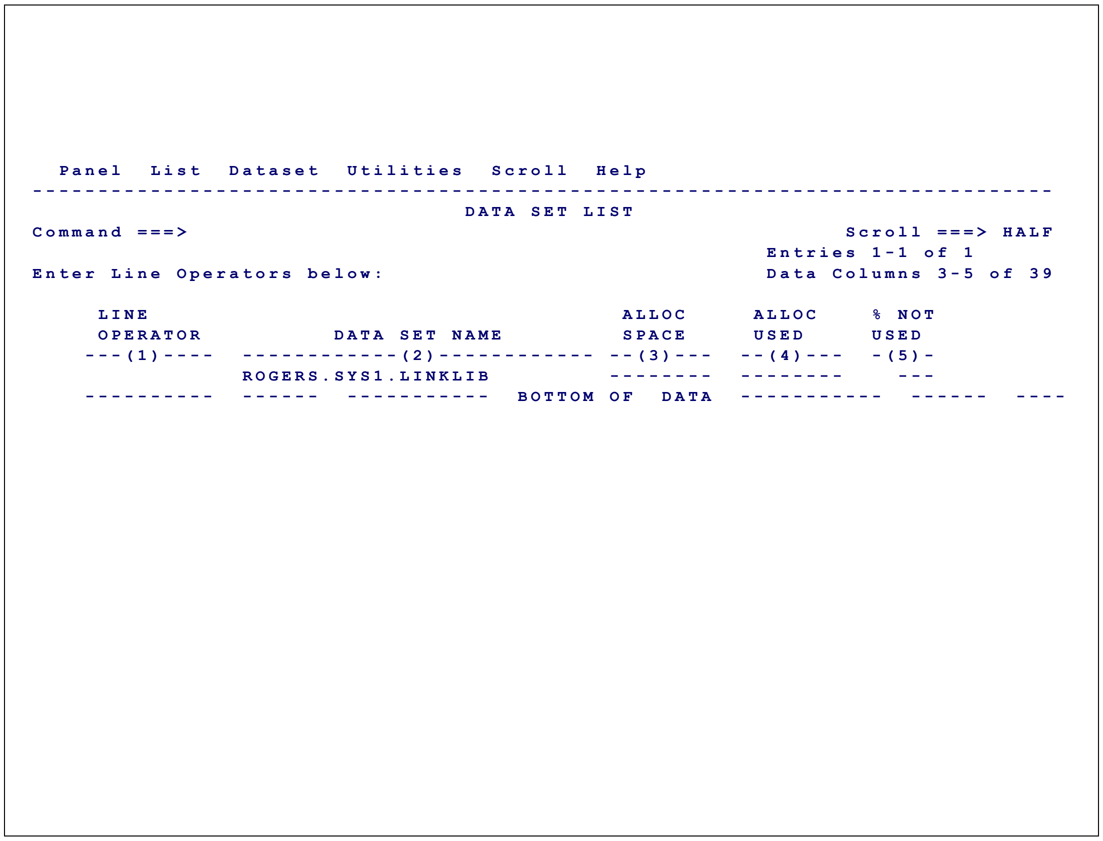

```
  Panel  List  Dataset  Utilities  Scroll  Help
------------------------------------------------------------------------------
                                DATA SET LIST
Command ===>                                                  Scroll ===> HALF
                                                        Entries 1-1 of 1
Enter Line Operators below:                             Data Columns 3-5 of 39

     LINE                                   ALLOC      ALLOC     % NOT
     OPERATOR          DATA SET NAME        SPACE      USED      USED
    ---(1)----  ------------(2)------------ --(3)---  --(4)---  -(5)-
                ROGERS.SYS1.LINKLIB          --------  --------    ---
    ----------  ------  -----------  BOTTOM OF  DATA  -----------  ------  ----
```

Figure 5-55   Saved ISMF Lists panel

ISMF lists

After obtaining a list (data set, data class, or storage class), you can save the list by typing **SAVE** *listname* in the Command panel field. To see the saved lists, use the option **L** (List) in the ISMF Primary Option Menu.

The List Application panel displays a list of all lists saved from ISMF applications. Each entry in the list represents a list that was saved. If there are no saved lists to be found, the ISMF Primary Option Menu panel is redisplayed with the message that the list is empty.

You can reuse and delete saved lists. From the List Application, you can reuse lists as though they were created from the corresponding application. You can then use line operators and commands to tailor and manage the information in the saved lists.

For more about the ISMF panel, refer to *z/OS DFSMS: Using the Interactive Storage Management Facility,* SC26-7411*.*

6

Catalogs

A *catalog* is a z/OS data set that describes other data set attributes and records the location of a data set so that the data set can be retrieved without requiring the user to specify its volume location. Multiple user catalogs contain information about user data sets, and a single master catalog contains entries for system data sets and user catalogs.

In z/OS, the component controlling catalogs is embedded in DFSMSdfp and is called Catalog Management. Catalog Management has one address space for itself named Catalog Address Space (CAS). This address space is used for buffering and to store control blocks, together with some code. The modern catalog structure in z/OS is named integrated catalog facility (ICF).

All data sets managed by the storage management subsystem (SMS) must be cataloged in an ICF catalog.

Most installations depend on the availability of catalog facilities to run production job streams and to support online users. For maximum reliability and efficiency, all permanent data sets should be cataloged and catalog recovery procedures must exist to guarantee continuous availability in z/OS.

6.1 Catalogs

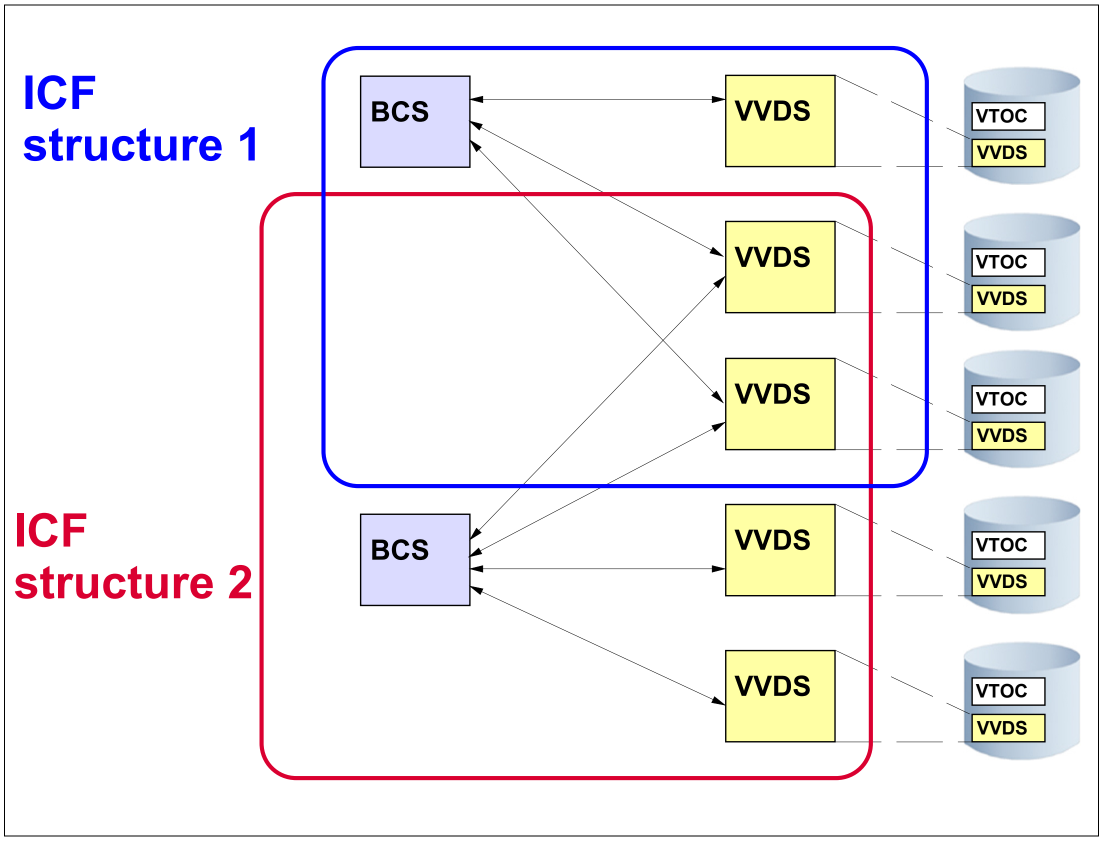


Figure 6-1 The ICF catalog structure

Catalogs

A catalog is a data set that contains information about other data sets. It provides users with the ability to locate a data set by name, without knowing the volume where the data set resides. When a data set catalog is utilized, your users need to know less about your storage setup. Thus, data sets can be moved from one device to another, without requiring a change in JCL DD statements that refer to an existing data set.

Cataloging data sets also simplifies backup and recovery procedures. Catalogs are the central information point for VSAM data sets; all VSAM data sets must be cataloged. In addition, all SMS-managed data sets must be cataloged.

Activity towards the catalog is much more intense in a Batch/TSO workload than in a CICS/DB2 workload, where the majority of data sets are allocated at CICS/DB2 initialization time.

The integrated catalog facility (ICF) catalog

An ICF consists of two different components, one basic catalog structure (BCS) and one or more VSAM volume data sets (VVDS).

When we talk about a catalog, we usually mean the BCS. The VVDS can be considered an extension of the volume table of contents (VTOC). The VVDS is volume-specific, whereas the complexity of the BCS depends on your definitions. The relationship between the BCS and the VVDS is many-to-many. That is, a BCS can point to multiple VVDSs and a VVDS can point to multiple BCSs.

6.2 The basic catalog structure (BCS)

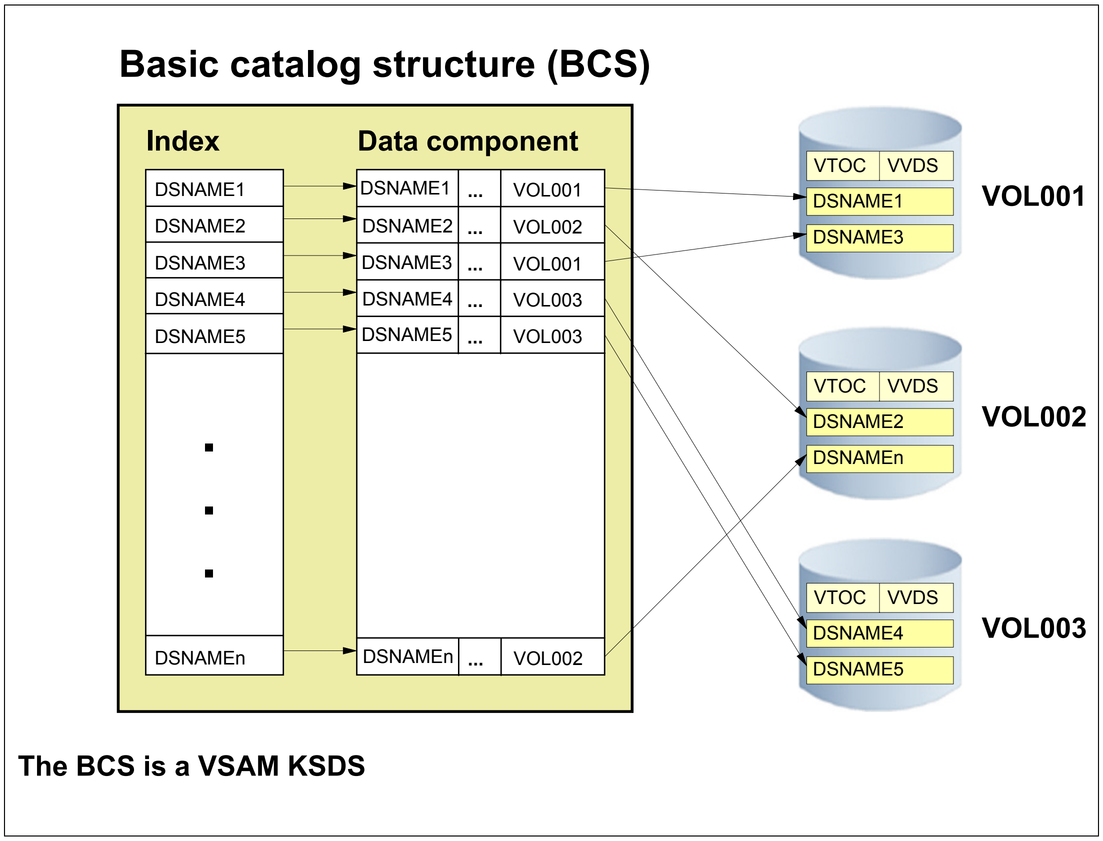


Figure 6-2 Basic catalog structure

Basic catalog structure (BCS)

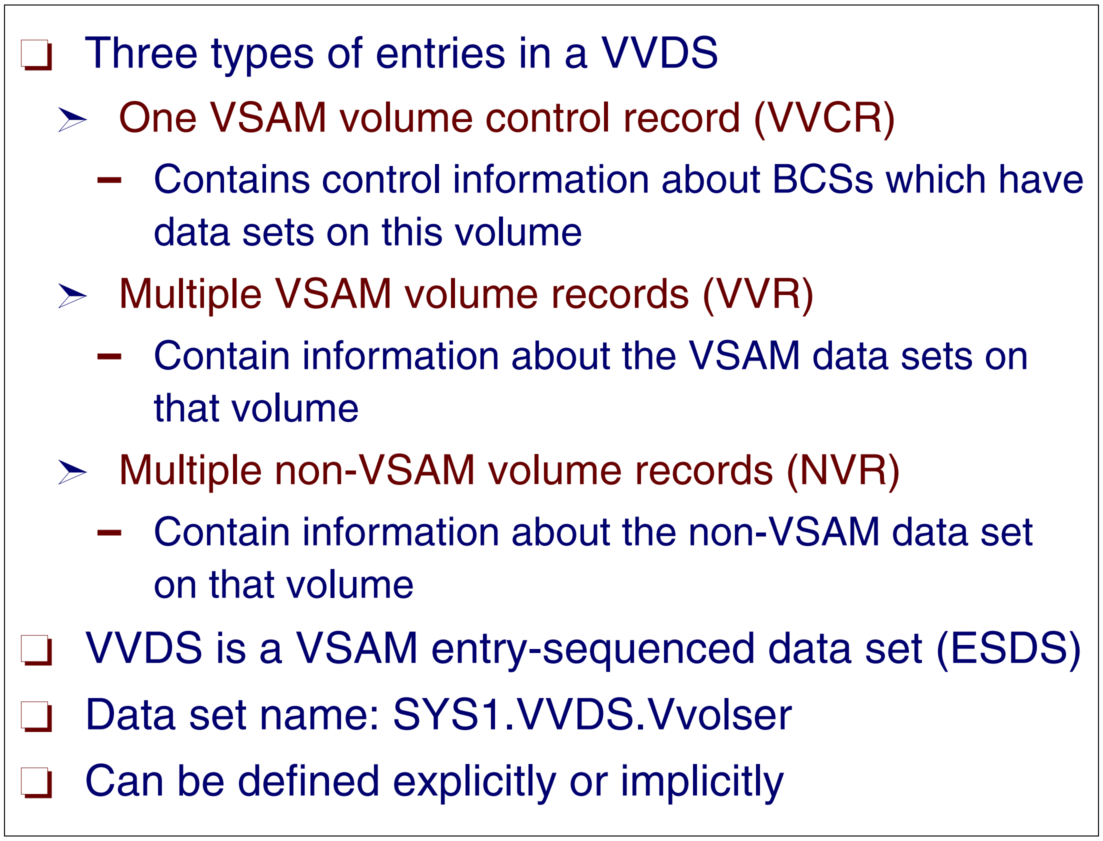
The basic catalog structure is a *VSAM key-sequenced data set* (KSDS). It uses the data set name as a key to store and retrieve data set information. For VSAM data sets, the BCS contains volume, security, ownership, and association information. For non-VSAM data sets, the BCS contains volume, ownership, and association information. In other words, the BCS portion of the ICF catalog contains the *static* information about the data set, the information that rarely changes.

For *non-VSAM* data sets that are *not SMS-managed*, all catalog information is contained within the BCS. For other types of data sets, there is other information available in the VVDS.

The BCS contains the information about where a data set resides. That can be a DASD volume or tape or an other storage medium.

Related information in the BCS is grouped into logical, variable-length, spanned records related by key. The BCS uses keys that are the *data set names* (plus one character for extensions). One control interval can contain multiple BCS records. To reduce the number of I/Os necessary for catalog processing, logically-related data is consolidated in the BCS.

6.3 The VSAM volume data set (VVDS)



- Three types of entries in a VVDS
  - One VSAM volume control record (VVCR)
    - Contains control information about BCSs which have data sets on this volume
  - Multiple VSAM volume records (VVR)
    - Contain information about the VSAM data sets on that volume
  - Multiple non-VSAM volume records (NVR)
    - Contain information about the non-VSAM data set on that volume
- VVDS is a VSAM entry-sequenced data set (ESDS)
- Data set name: SYS1.VVDS.Vvolser
- Can be defined explicitly or implicitly

Figure 6-3 The VSAM volume data set

The VSAM volume data set (VVDS)

The VSAM volume data set (VVDS) contains additional catalog information (not contained in the BCS) about the VSAM and SMS-managed non-VSAM data sets residing on the volume where the VVDS is located. Every volume containing any VSAM or any SMS-managed data sets must have a VVDS on it. The VVDS acts as sort of a VTOC extension for certain types of data sets. A VVDS may have data set information about data sets cataloged in distinct BCSs.

Entry types in the VVDS

There are three types of entries in a VVDS. They are:

- VSAM volume control records (VVCR)
  - First logical record in a VVDS
  - Contain information for management of DASD space and the names of the BCSs that have data sets on the volume
- VSAM volume records (VVR)
  - Contain information about a VSAM data set residing on the volume
  - Number of VVRs varies according to the type of data set and the options specified for the data set
  - Also included are data set characteristics, SMS data, extent information
  - There is one VVR describing the VVDs itself

- Non-VSAM volume record (NVR)
  - Equivalent to a VVR for SMS-managed non-VSAM data sets
  - Contains SMS-related information

VVDS characteristics

The VVDS is a VSAM entry-sequenced data set (ESDS) that has a 4 KB control interval size. The hexadecimal RBA of a record is used as its key or identifier.

A VVDS is recognized by the restricted data set name:

SYS1.VVDS.Vvolser

Volser is the volume serial number of the volume on which the VVDS resides.

You can *explicitly* define the VVDS using IDCAMS, or it is *implicitly* created after you define the first VSAM or SMS-managed data set on the volume.

VVDSSPACE keyword

Before z/OS V1R7, the default space parameter is TRACKS(10,10), which may be too small for sites that use custom 3390 volumes (the ones greater than 3390-9). With z/OS V1R7, there is a new VVDSSPACE keyword of the **F CATALOG** command, as follows:

```
F CATALOG,VVDSSPACE(primary,secondary)
```

An explicitly defined VVDS is not related to any BCS until a data set or catalog object is defined on the volume. As data sets are allocated on the VVDS volume, each BCS with VSAM data sets or SMS-managed data sets residing on that volume is related to the VVDS.

VVDSSPACE indicates that the Catalog Address Space should use the values specified as the primary and secondary allocation amount in tracks for an implicitly defined VVDS. The default value is ten tracks for both the primary and secondary values. The specified values are preserved across a Catalog Address Space restart, but are not preserved across an IPL.

6.4 Catalogs by function

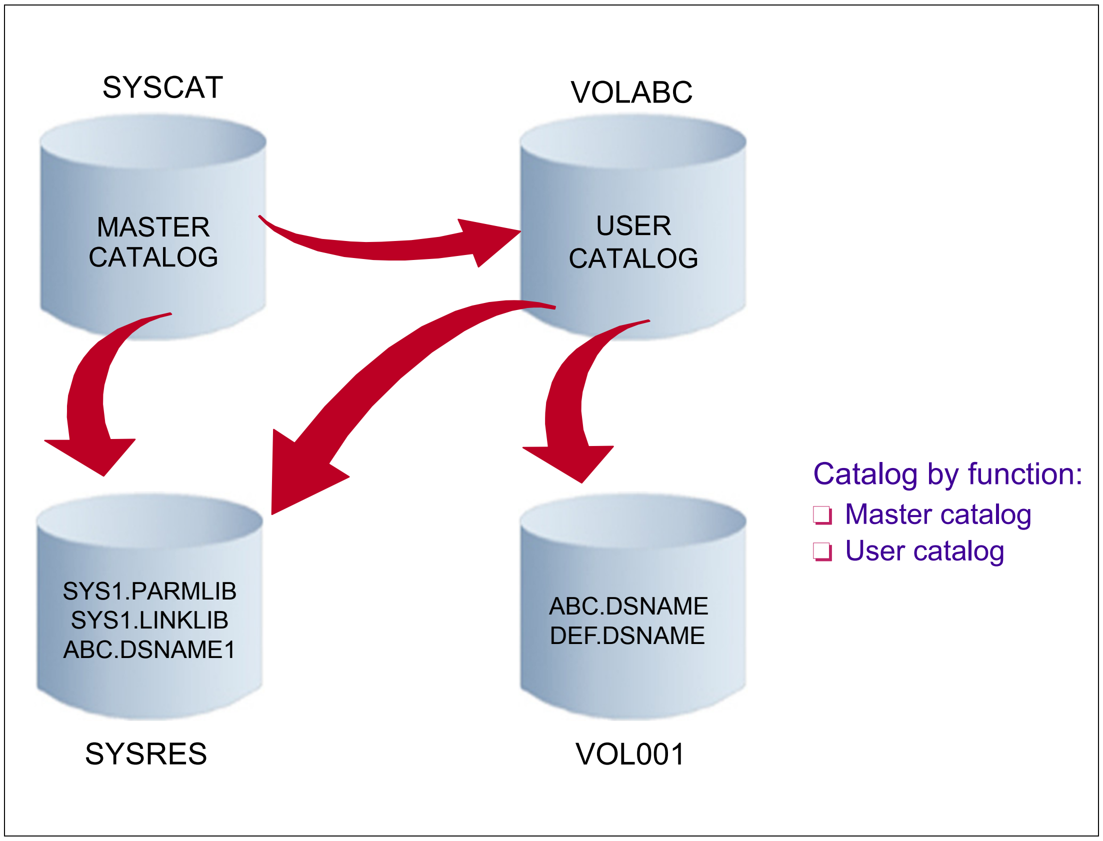


Figure 6-4 The master catalog

Catalogs by function

By function, the catalogs (BCSs) can be classified as *master* catalog and *user* catalog. A particular case of a user catalog is the *volume* catalog, which is a user catalog containing only tape library and tape volume entries.

There is *no* structural difference between a master catalog and a user catalog. What makes a master catalog different is how it is used, and what data sets are cataloged in it. For example, the same catalog can be master in one z/OS and user in the other z/OS.

The master catalog

Each system has *one* active master catalog. One master catalog can be shared between different MVS images. It does not have to reside on the system residence volume (the one that is IPLed).

The master catalog for a system must contain entries for all user catalogs and their aliases that the system uses. Also, all SYS1 data sets must be cataloged in the master catalog for proper system initialization.

Attention: To minimize update activity to the master catalog, and to reduce the exposure to breakage, we strongly recommend that *only* SYS1 data sets, user catalog connector records, and the aliases pointing to those connectors should be in the master catalog.

During a system initialization, the master catalog is read so that system data sets and catalogs can be located.

Identifying the master catalog for IPL

At IPL, you must indicate the location (volser and data set name) of the master catalog. This information can be specified in one of two places:

- SYS1.NUCLEUS member SYSCATxx (default is SYSCATLG)
- SYS1.PARMLIB/SYSn.IPLPARM member LOADxx. This method is recommended.

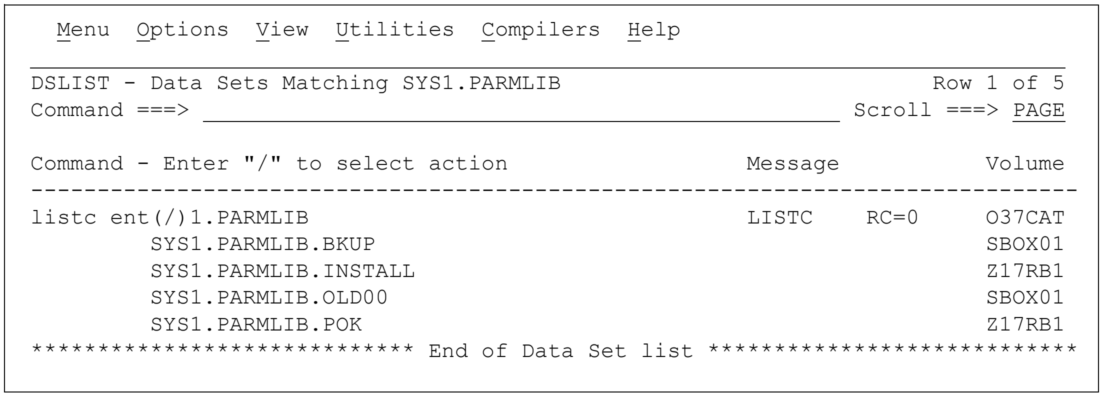
For more information refer to *z/OS MVS Initialization and Tuning Reference,* SA22-7592.

Determine the master catalog on a running system

You can use the IDCAMS **`LISTCAT`** command for a data set with a high-level qualifier (HLQ) of *SYS1* to determine the master catalog on a system. Since all data sets with an HLQ of SYS1 should be in the master catalog, the catalog shown in the LISTCAT output is the master catalog.

For information about the IDCAMS **`LISTCAT`** command, see also "Listing a catalog" on page 322.

If you do not want to run an IDCAMS job, you can run LISTCAT as a line command in ISPF Option 3.4. List the SYS1.PARMLIB and type **`LISTC ENT(/)`**, as shown in Figure 6-5.



```
  Menu  Options  View  Utilities  Compilers  Help
 ------------------------------------------------------------------------------
DSLIST - Data Sets Matching SYS1.PARMLIB                            Row 1 of 5
Command ===>                                                  Scroll ===> PAGE

Command - Enter "/" to select action                  Message          Volume
-------------------------------------------------------------------------------
listc ent(/)1.PARMLIB                                 LISTC    RC=0    O37CAT
         SYS1.PARMLIB.BKUP                                             SBOX01
         SYS1.PARMLIB.INSTALL                                          Z17RB1
         SYS1.PARMLIB.OLD00                                            SBOX01
         SYS1.PARMLIB.POK                                              Z17RB1
***************************** End of Data Set list ****************************
```

Figure 6-5 Example for a LISTCAT in ISPF option 3.4

Note: The **/** specifies to use the data set name on the line where the command is entered.

The output of this command will be something like:

```
NONVSAM ------- SYS1.PARMLIB
     IN-CAT --- MCAT.SANDBOX.Z17.SBOX00
```

User catalogs

The difference between the master catalog and the user catalogs, as we saw, is in the *function*. User catalogs should be used to contain information about your installation cataloged data sets other than SYS1 data sets. There are no set rules as to how many you should have or how large they should be. It depends entirely on your environment. Cataloging data sets for two unrelated applications in the same catalog creates a single point of failure for them that otherwise may not exist. An assessment of the impact of outage of a given catalog may help determine if it is too big or would impact too many different applications.

6.5 Using aliases

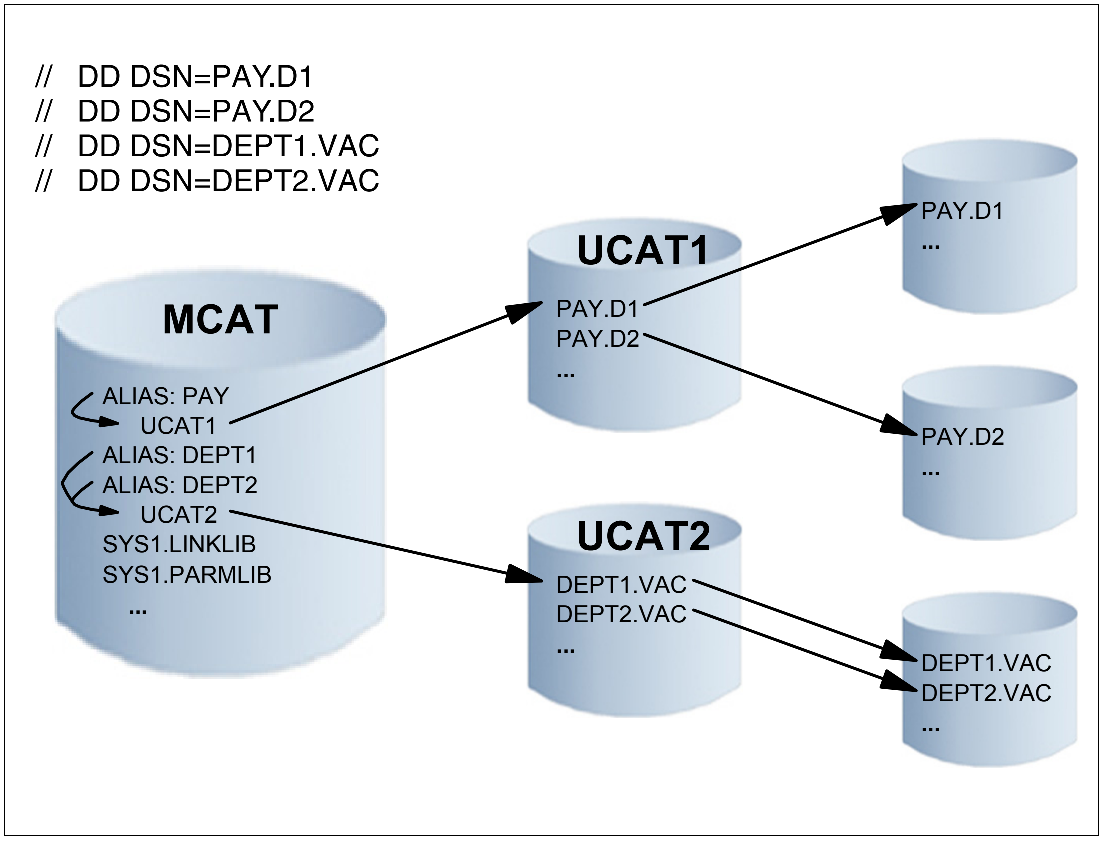


Figure 6-6 Using aliases

Using aliases

Aliases are used to tell catalog management which user catalog your data set is cataloged in. First, you place a pointer to an user catalog in the master catalog through the IDCAMS **`DEFINE UCAT`** command. Then, you define an appropriate alias name for a user catalog in the master catalog. Next, match the high-level qualifier (HLQ) of your data set with the alias. This identifies the appropriate user catalog to be used to satisfy the request.

In Figure 6-6, all data sets with an HLQ of PAY have their information in the user catalog UCAT1 because in the master catalog there is an alias PAY pointing to UCAT1.

The data sets with an HLQ of DEPT1 and DEPT2 respectively have their information in the user catalog UCAT2 because in the master catalog there are aliases DEPT1 and DEPT2 pointing to UCAT2.

Note: Aliases can also be used with non-VSAM data sets in order to create alternate names to the same data set. Those aliases are *not* related to a user catalog.

To define an alias, use the IDCAMS command **`DEFINE ALIAS`**. An example is in “Defining a catalog and its aliases” on page 316.

Multilevel aliases

You can augment the standard catalog search order by defining *multilevel* catalog aliases. A multilevel catalog alias is an alias of two or more high-level qualifiers. You can define aliases of up to four high-level qualifiers.

However, the multilevel alias facility should only be used when a better solution cannot be found. The need for the multilevel alias facility may indicate some data set naming conventions problems.

For more information about the multilevel alias facility refer to *z/OS DFSMS: Managing Catalogs,* SC26-7409.

6.6 Catalog search order

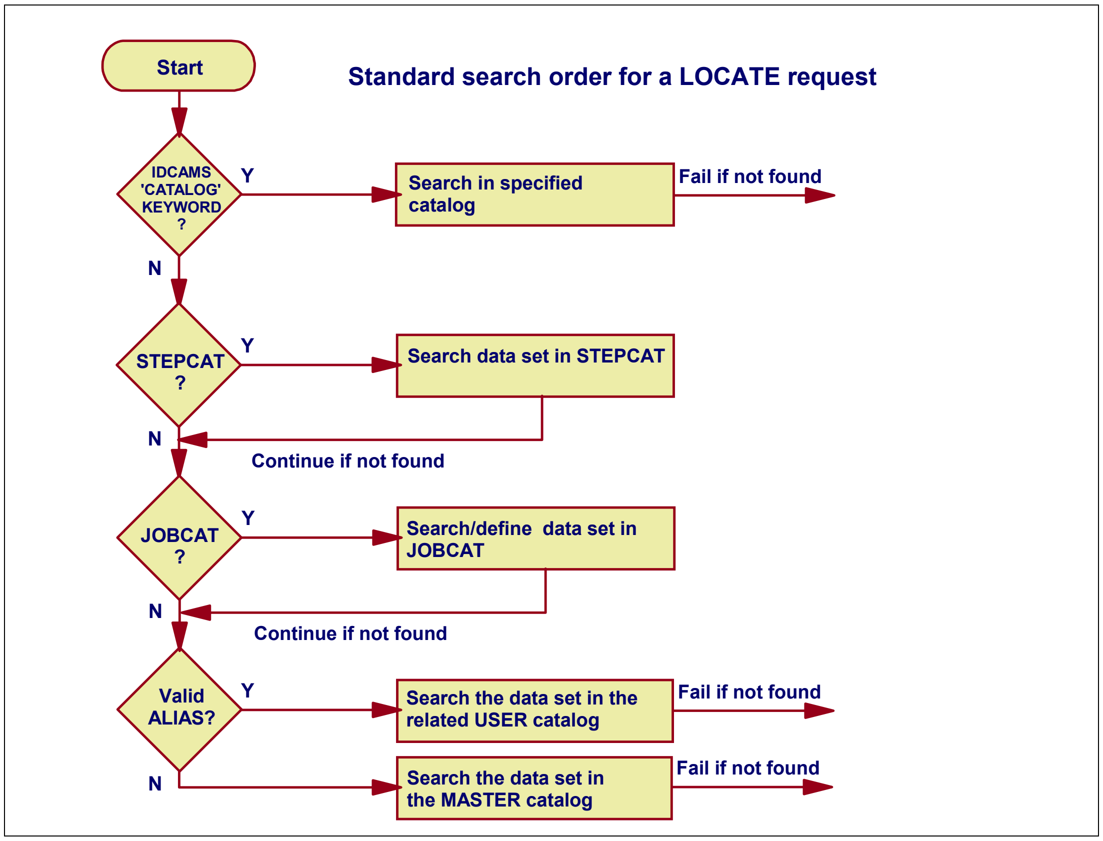

Start
Standard search order for a LOCATE request
IDCAMS 'CATALOG' KEYWORD ?
Y
N
Search in specified catalog
Fail if not found
STEPCAT ?
Y
N
Search data set in STEPCAT
Continue if not found
JOBCAT ?
Y
N
Search/define data set in JOBCAT
Continue if not found
Valid ALIAS?
Y
N
Search the data set in the related USER catalog
Fail if not found
Search the data set in the MASTER catalog
Fail if not found

Figure 6-7 Catalog search order for a LOCATE request

Catalog search order

Locate is an SVC that calls catalog management asking for a data set name search. Most catalog searches should be based on catalog aliases. Some alternatives to catalog aliases are available for directing a catalog request, specifically the JOBCAT and STEPCAT DD statements, the CATALOG parameter of access method services, and the name of the catalog. JOBCAT and STEPCAT are no longer allowed beginning with z/OS V1R7.

Search order for catalogs for a data set define request

For the system to determine where a data set is to be cataloged, the following search order is used to find the catalog:

1. If the IDCAMS DEFINE (creation and cataloging) statement is used and the CATALOG parameter is specified, then use the catalog that is referred to.
2. Otherwise, use the catalog named in the STEPCAT DD statement.
3. If no STEPCAT, use the catalog named in the JOBCAT DD statement.
4. If no JOBCAT, and the HLQ is a catalog alias, use the catalog identified by the alias or the catalog whose name is the same as the HLQ of the data set.
5. If no catalog has been identified yet, use the master catalog.

When you specify a catalog in the IDCAMS CATALOG parameter, and you have appropriate RACF authority to the FACILITY class profile STGADMIN.IGG.DIRCAT, the catalog you specify is used. For instance:

```
   DEFINE CLUSTER (NAME(PROD.PAYROLL) CATALOG(SYS1.MASTER.ICFCAT))
```

defines the data set PROD.PAYROLL in catalog SYS1.MASTER.ICFCAT.

You can use RACF to prevent the use of the CATALOG parameter and restrict the ability to define data sets in the master catalog.

Search order for locating a data set

The following search order is used to locate the catalog for an already cataloged data set:

1. If at any IDCAMS invocation there is a need to locate a data set and the CATALOG parameter is specified, use the catalog that is referred to; if not found, fail the job.
2. Otherwise, search all catalogs specified in a STEPCAT DD statement in order.
3. If not found, search all catalogs specified in a JOBCAT DD statement in order.
4. If not found, and the HLQ is an alias for a catalog, search this user catalog; or if the HLQ is the name of a catalog, search this catalog. If not found, fail the job.
5. Otherwise, search the master catalog.

Note: For SMS-managed data sets, JOBCAT and STEPCAT DD statements are not allowed and cause a job failure. Also, they are *not recommended* even for non-SMS data sets, because they may cause conflicted information. Therefore, do not use them, and remember that since z/OS V1R7 they are phased out.

To use an alias to identify the catalog to be searched, the data set must have more than one data set qualifier.

For information about the catalog standard search order also refer to *z/OS DFSMS: Managing Catalogs,* SC26-7409.

6.7 Defining a catalog and its aliases

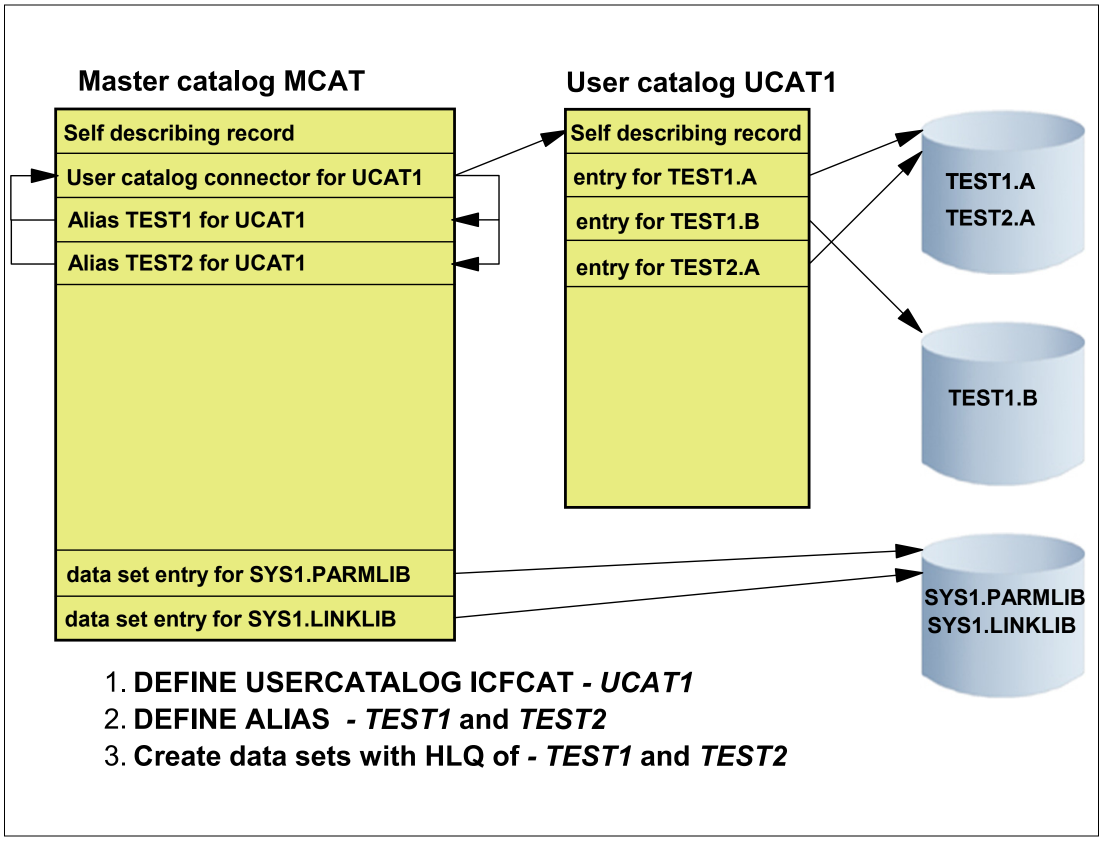


Figure 6-8 JCL to create a basic catalog structure

Defining a catalog

You can use the IDCAMS to define and maintain catalogs. See also “Access method services (IDCAMS)” on page 135. Defining a master catalog or user catalog is basically the same.

```
//DEFCAT   JOB  ...
    //DEFCAT  EXEC PGM=IDCAMS
    //SYSPRINT DD SYSOUT=A
    //SYSIN    DD *
      DEFINE USERCATALOG                -
            ( NAME(OTTO.CATALOG.TEST) -
              MEGABYTES(15 15)          -
              VOLUME(VSF6S4)            -
              ICFCATALOG                -
              FREESPACE(10 10)          -
              STRNO(3)  )               -
           DATA( CONTROLINTERVALSIZE(4096) -
                 BUFND(4)  )               -
           INDEX( BUFNI(4)  )
    /*
```

Figure 6-9 Sample JCL to define a BCS

The example in Figure 6-9 shows the JCL you can use to define a user catalog. The catalog defined, OTTO.CATALOG.TEST, is placed on volume VSF6S4, and is allocated with 15 megabytes primary and secondary space.

Use the access method services command **DEFINE USERCATALOG ICFCATALOG** to define the basic catalog structure (BCS) of an ICF catalog. Using this command you do not specify whether you want to create a user or a master catalog. How to identify the master catalog to the system is described in "Catalogs by function" on page 310.

A connector entry to this user catalog is created in the master catalog, as the listing in Figure 6-10 shows.

```
                              LISTING FROM CATALOG -- CATALOG.MVSICFM.VVSF6C1
USERCATALOG --- OTTO.CATALOG.TEST
     HISTORY
       RELEASE----------------2
     VOLUMES
        VOLSER------------VSF6S4      DEVTYPE------X'3010200F'
VOLFLAG------------PRIME
     ASSOCIATIONS--------(NULL)
```

Figure 6-10   Listing of the user catalog connector entry

The attributes of the user catalog are not defined in the master catalog. They are described in the user catalog itself and its VVDS entry. This is called the *self describing record*. The self describing record is given a key of binary zeros to ensure it is the first record in the catalog. There are no associations (aliases) yet for this user catalog. To create any, you need to define aliases.

To define a volume catalog (for tapes), use the parameter **VOLCATALOG** instead of **ICFCATALOG**. See *z/OS DFSMS Access Method Services for Catalogs*, SC26-7394, for more detail.

Defining a BCS with a model

When you define a BCS or VVDS, you can use an existing BCS or VVDS as a model for the new one. The attributes of the existing data set are copied to the newly defined data set unless you explicitly specify a different value for an attribute. You can override any of a model's attributes.

If you do not want to change or add any attributes, you need only supply the entry name of the object being defined and the MODEL parameter. When you define a BCS, you must also specify the volume and space information for the BCS.

For further information about using a model refer to *z/OS DFSMS: Managing Catalogs,* SC26-7409.

Defining aliases

In order to use a catalog, the system must be able to determine which data sets should be defined in that catalog. The simplest way to accomplish this is to define aliases in the master catalog for the user catalog. Before defining an alias, carefully consider the effect the new alias has on old data sets. A poorly chosen alias could make some data sets inaccessible.

You can define aliases for the user catalog in the same job in which you define the catalog by including **DEFINE ALIAS** commands after the **DEFINE USERCATALOG** command. You can use conditional operators to ensure the aliases are only defined if the catalog is successfully defined. After the catalog is defined, you can add new aliases or delete old aliases.

Catalog aliases are defined only in the master catalog, which contains an entry for the user catalog. The number of aliases a catalog can have is limited by the maximum record size for the master catalog.

You cannot define an alias if a data set cataloged in the master catalog has the same high-level qualifier as the alias. The **DEFINE ALIAS** command fails with a "`Duplicate data set name`" error. For example, if a catalog is named TESTE.TESTSYS.ICFCAT, you cannot define the alias TESTE for any catalog.

Use the sample SYSIN for an IDCAMS job in Figure 6-11 to define aliases TEST1 and TEST2.

```
  DEFINE ALIAS          -
    (NAME(TEST1)        -
     RELATE(OTTO.CATALOG.TEST))
  DEFINE ALIAS          -
    (NAME(TEST2)        -
     RELATE(OTTO.CATALOG.TEST))
```

Figure 6-11 DEFINE ALIAS example

These definitions result in the following entries in the master catalog (Figure 6-12).

```
ALIAS --------- TEST1
     IN-CAT --- CATALOG.MVSICFM.VVSF6C1
     HISTORY
       RELEASE----------------2
     ASSOCIATIONS
       USERCAT--OTTO.CATALOG.TEST
...
ALIAS --------- TEST2
     IN-CAT --- CATALOG.MVSICFM.VVSF6C1
     HISTORY
       RELEASE----------------2
     ASSOCIATIONS
       USERCAT--OTTO.CATALOG.TEST
```

Figure 6-12 Listing an ALIAS in the master catalog

Both aliases have an association to the newly defined user catalog. If you now create a new data set with an HLQ of TEST1 or TEST2, its entry will be directed to the new user catalog. Also, the listing of the user catalog connector now shows both aliases (Figure 6-13).

```
USERCATALOG --- OTTO.CATALOG.TEST
     HISTORY
       RELEASE----------------2
     VOLUMES
       VOLSER------------VSF6S4     DEVTYPE------X'3010200F'
VOLFLAG------------PRIME
     ASSOCIATIONS
       ALIAS----TEST1
       ALIAS----TEST2
```

Figure 6-13 Listing of the user catalog connector entry

6.8 Using multiple catalogs

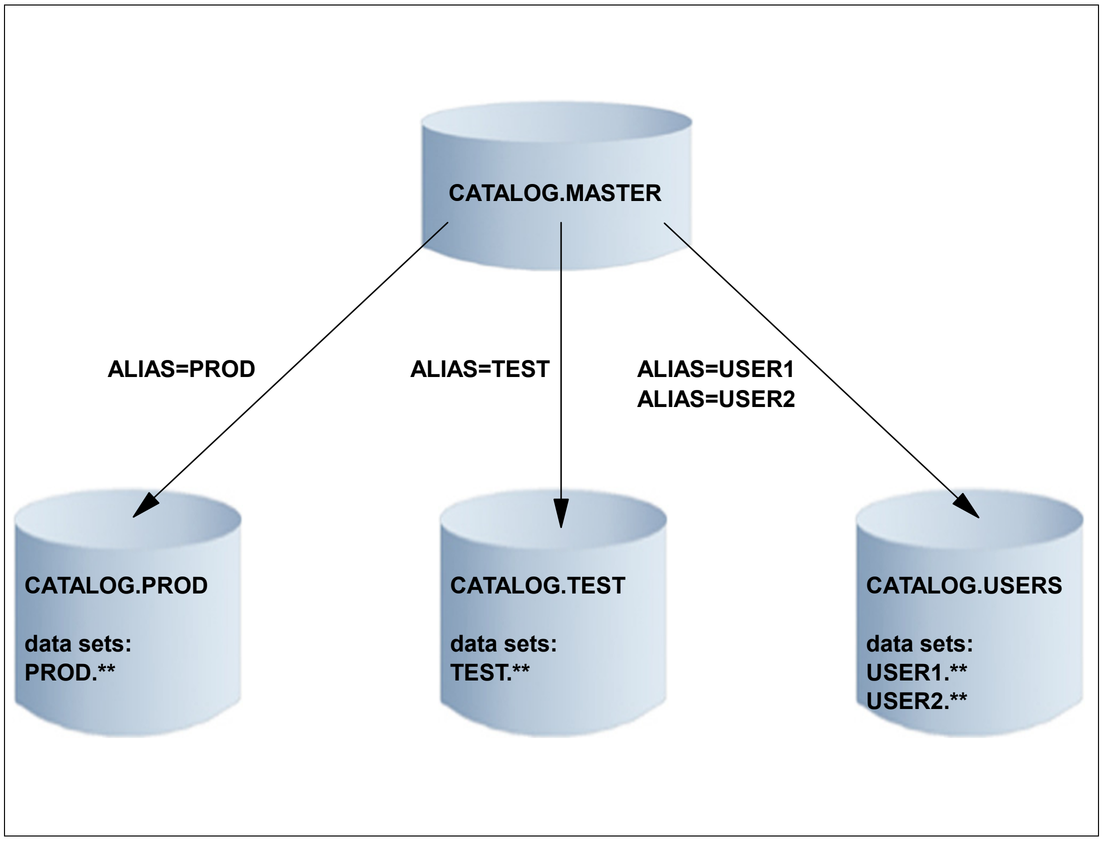


Figure 6-14 Using multiple catalogs

Using multiple catalogs

Multiple catalogs on multiple volumes may perform better than fewer catalogs on fewer volumes. This is because of the interference between requests to the same catalog; for example, a single shared catalog being locked out by another system in the sysplex. This situation can occur if some other application issues a RESERVE against the volume that has nothing to do with catalog processing. Another reason can be that there is more competition to use the available volumes, and thus more I/O can be in progress concurrently.

Tip: Convert all intra-sysplex RESERVES in global ENQs through the conversion RNL.

Independent of the number of catalogs, use the virtual lookaside facility (VLF) for buffering the user catalog CIs. The master catalog CIs are naturally buffered in the catalog address space (CAS). Multiple catalogs can reduce the impact of the loss of a catalog by:

- Reducing the time necessary to recreate any given catalog
- Allowing multiple catalog recovery jobs to be in process at the same time

Recovery from a pack failure is dependent on the total amount of catalog information on a volume—regardless of whether this information is stored in one or many catalogs.

When you are using multiple user catalogs you should consider a concept of grouping data sets under different high-level qualifiers. You can then spread them over multiple catalogs by defining aliases for the different catalogs.

6.9 Sharing catalogs across systems

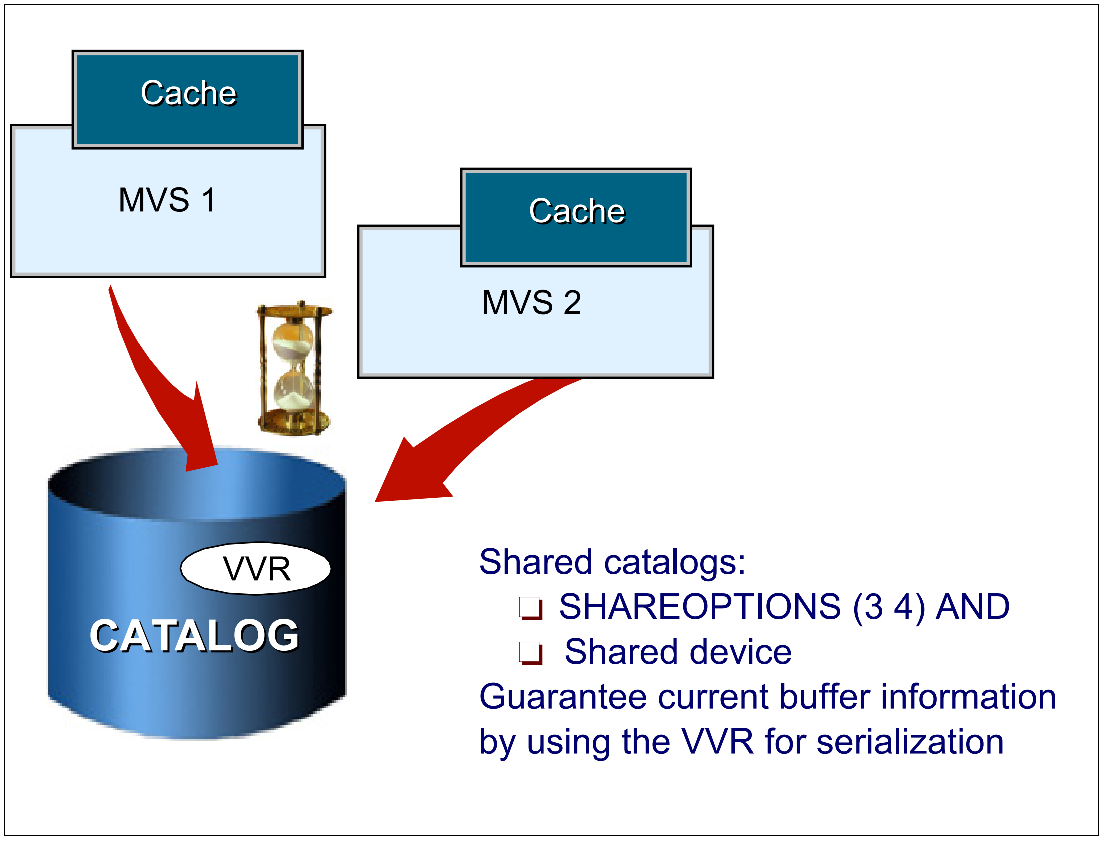


Figure 6-15 Sharing catalogs

Sharing catalogs across systems

A shared catalog is a basic catalog structure (BCS) that is eligible to be used by more than one system. It must be defined with SHAREOPTIONS(3 4), and reside on a shared volume. A DASD volume is initialized as shared using the MVS hardware configuration definition (HCD) facility.

Note: The device must be defined as shared to all systems that access it.

If some systems have the device defined as shared and some do not, catalog corruption will occur. Check with your system programmer to determine shared volumes. Note that it is not necessary that the catalog actually be shared between systems; the catalog address space assumes it is shared if it meets the criteria stated. All VVDSs are defined as shared. Tape volume catalogs can be shared in the same way as other catalogs.

By default, catalogs are defined with SHAREOPTIONS(3 4). You can specify that a catalog is not to be shared by defining the catalog with SHAREOPTIONS(3 3). Only define a catalog as unshared if you are certain it will not be shared. Place unshared catalogs on volumes that have been initialized as unshared. Catalogs that are defined as unshared and that reside on shared volumes will become damaged if referred to by another system.

If you need to share data sets across systems, it is advisable that you share the catalogs that contain these data sets. A BCS catalog is considered shared when *both* of the following are true:

- It is defined with SHAREOPTIONS (3 4).
- It resides on a shared device, as defined at HCD.

Attention: To avoid catalog corruption, define a catalog volume on a shared UCB and set catalog SHAREOPTIONS to (3 4) on *all* systems sharing a catalog.

Using SHAREOPTIONS 3 means that VSAM does not issue the ENQ SYSVSAM SYSTEMS for the catalog; SHAREOPTIONS 4 means that the VSAM buffers need to be refreshed.

You can check whether a catalog is shared by running the operator command:

```
MODIFY CATALOG,ALLOCATED
```

There is a flag in the catalog that indicates whether the catalog is shared.

If a catalog is not really shared with another system, move the catalog to an unshared device or alter its SHAREOPTIONS to (3 3). To prevent potential catalog damage, never place a catalog with SHAREOPTIONS (3 3) on a shared device.

There is one VVR in a shared catalog that is used as a log by all the catalog management accessing such catalog. This log is used to guarantee the coherency of each catalog buffer in each z/OS system.

Guaranteeing current information

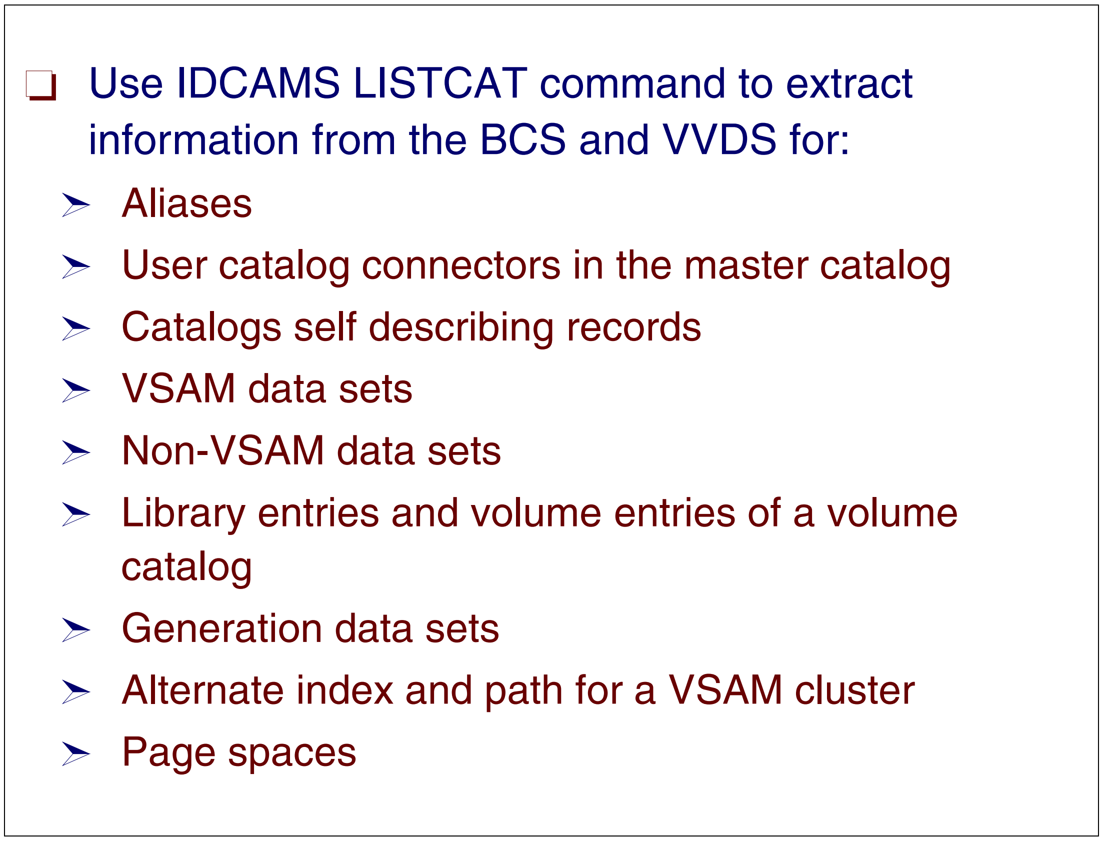
For performance reasons, CIs contained in a master catalog are buffered in the catalog address space (CAS). Optionally (and highly recommended), the user catalogs are buffered in VLF data space, so before each search in the buffers (looking for a match) a simple I/O operation checking is performed in the log VVR. If the catalog was updated by another system, the information in the buffer needs to be refreshed by reading the data from the volume.

The checking also affects performance because in order to maintain the integrity, for every catalog access, a special VVR in the shared catalog must be read before using the cached version of the BCS record. This access implies a DASD reserve and I/O operations.

To avoid I/O operations to read the VVR you can use enhanced catalog sharing (ECS). For information about ECS, see “Enhanced catalog sharing” on page 351.

Checking also ensures that the control blocks for the catalog in the CAS are updated. This occurs in the event the catalog has been extended, or otherwise altered from another system. This checking maintains data integrity.

6.10 Listing a catalog



- Use IDCAMS LISTCAT command to extract information from the BCS and VVDS for:
  - Aliases
  - User catalog connectors in the master catalog
  - Catalogs self describing records
  - VSAM data sets
  - Non-VSAM data sets
  - Library entries and volume entries of a volume catalog
  - Generation data sets
  - Alternate index and path for a VSAM cluster
  - Page spaces

Figure 6-16 Listing a catalog

Requesting information from a catalog

You can list catalog records using the IDCAMS **LISTCAT** command, or the ISMF line operator command **CATLIST**. **CATLIST** produces the same output as **LISTCAT**. With z/OS V1R8, the **LISTCAT** processing has performance improvements.

You can use the **LISTCAT** output to monitor VSAM data sets including catalogs. The statistics and attributes listed can be used to help determine if you should reorganize, recreate, or otherwise alter a VSAM data set to improve performance or avoid problems.

The **LISTCAT** command can be used in many variations to extract information about a particular entry in the catalog. It extracts the data from the BCS and VVDS.

LISTCAT examples

Here are some **LISTCAT** examples you should know to monitor your catalogs:

- List all ALIAS entries in the master catalog:

  ```
  LISTCAT ALIAS CAT(master.catalog.name)
  ```

  This command provides a list of all aliases that are currently defined in your master catalog. If you need information only about one specific alias, use the keyword **ENTRY(aliasname)** and specify **ALL** to get detailed information. For a sample output of this command see Figure 6-12 on page 318.

- List a user catalog connector in the master catalog:

  ```
  LISTCAT ENT(user.catalog.name) ALL
  ```

  You can use this command to display the volser and the alias associations of a user catalog as it is defined in the master catalog. For a sample output of this command see Figure 6-13 on page 318.

- List the catalog's self describing record:

  ```
  LISTCAT ENT(user.catalog.name) CAT(user.catalog.name) ALL
  ```

  This gives you detailed information about a user catalog, like attributes, statistics, extent information, and more. Because the self describing record is in the user catalog, you need to specify the name of the user catalog in the **`CAT`** statement. If you don't use the **`CAT`** keyword, only the user catalog connector information from the master catalog is listed as in the previous example. Figure 6-17 shows a sample output for this command.

```
                             LISTING FROM CATALOG -- CATALOG.MVSICFU.VVSF6C1
CLUSTER ------- 00000000000000000000000000000000000000000000
     HISTORY
       DATASET-OWNER-----(NULL)     CREATION--------2004.260
       RELEASE----------------2     EXPIRATION------0000.000
       BWO STATUS--------(NULL)     BWO TIMESTAMP-----(NULL)
       BWO---------------(NULL)
     PROTECTION-PSWD-----(NULL)     RACF----------------(NO)
     ASSOCIATIONS
       DATA-----CATALOG.MVSICFU.VVSF6C1
       INDEX----CATALOG.MVSICFU.VVSF6C1.CATINDEX
   DATA ------- CATALOG.MVSICFU.VVSF6C1
     HISTORY
       DATASET-OWNER-----(NULL)     CREATION--------2004.260
       RELEASE----------------2     EXPIRATION------0000.000
       ACCOUNT-INFO-----------------------------------(NULL)
     PROTECTION-PSWD-----(NULL)     RACF----------------(NO)
     ASSOCIATIONS
       CLUSTER--00000000000000000000000000000000000000000000
     ATTRIBUTES
       KEYLEN----------------45     AVGLRECL------------4086     BUFSPACE-----------11776     CISIZE--------------4096
       RKP--------------------9     MAXLRECL-----------32400     EXCPEXIT----------(NULL)     CI/CA----------------180
       BUFND------------------4     STRNO------------------3
       SHROPTNS(3,4)      SPEED     UNIQUE           NOERASE     INDEXED        NOWRITECHK     NOIMBED       NOREPLICAT
       UNORDERED        NOREUSE     SPANNED        NOECSHARE     ICFCATALOG
     STATISTICS
       REC-TOTAL--------------0     SPLITS-CI-------------34     EXCPS------------------0
       REC-DELETED--------11585     SPLITS-CA--------------0     EXTENTS----------------1
       REC-INSERTED-----------0     FREESPACE-%CI---------10     SYSTEM-TIMESTAMP:
       REC-UPDATED------------0     FREESPACE-%CA---------10          X'0000000000000000'
       REC-RETRIEVED----------0     FREESPC----------3686400
     ALLOCATION
       SPACE-TYPE------CYLINDER     HI-A-RBA---------3686400
       SPACE-PRI--------------5     HI-U-RBA----------737280
       SPACE-SEC--------------5
     VOLUME
       VOLSER------------VSF6C1     PHYREC-SIZE---------4096     HI-A-RBA---------3686400     EXTENT-NUMBER----------1
       DEVTYPE------X'3010200F'     PHYRECS/TRK-----------12     HI-U-RBA----------737280     EXTENT-TYPE--------X'00'
       VOLFLAG------------PRIME     TRACKS/CA-------------15
       EXTENTS:
       LOW-CCHH-----X'00110000'     LOW-RBA----------------0     TRACKS----------------75
       HIGH-CCHH----X'0015000E'     HIGH-RBA---------3686399
   INDEX ------ CATALOG.MVSICFU.VVSF6C1.CATINDEX
...
```

Figure 6-17 Example of a LISTCAT output for a user catalog

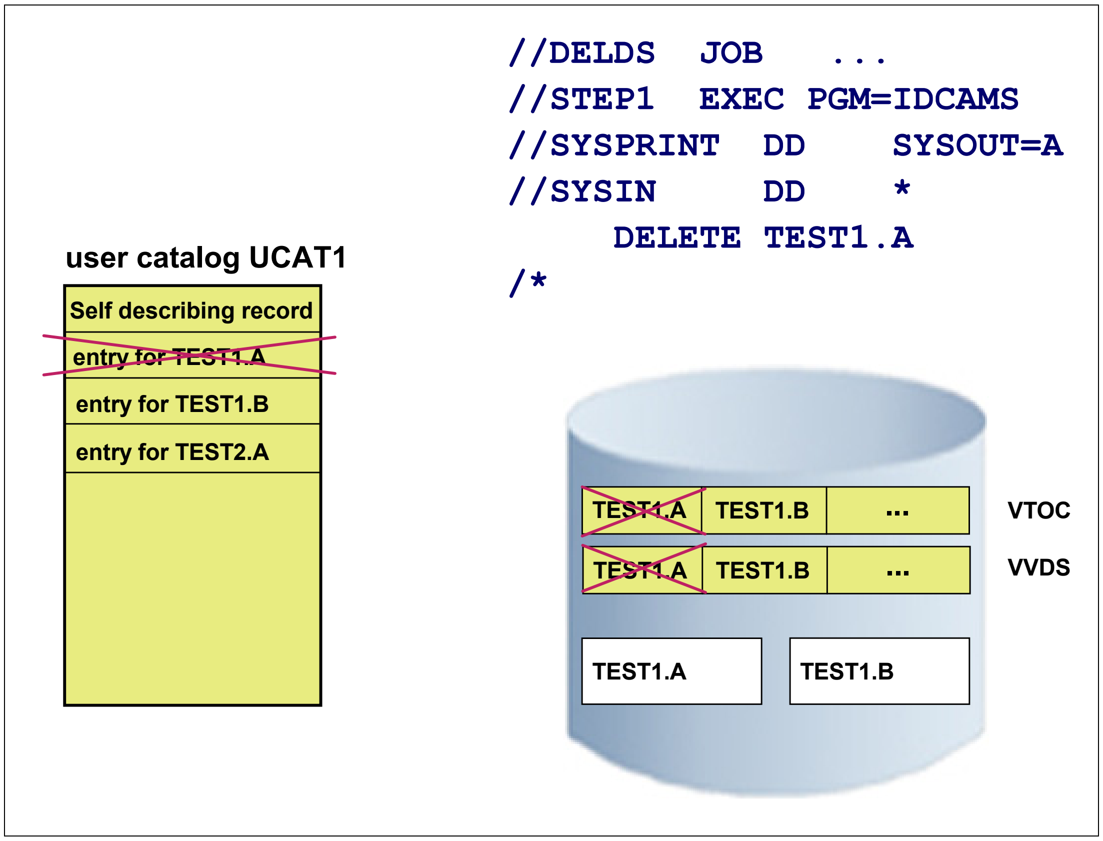
- Listing a VSAM or non-VSAM data set:

  ```
  LISTCAT ENT(data.set.name) ALL
  ```

  The output for a VSAM data set looks the same as in Figure 6-17 (remember: a catalog is a VSAM data set). For a non-VSAM data set the output is much shorter.

  You can use the **`LISTCAT`** command to list information for other catalog entries as well. For information about **`LISTCAT`**, see *z/OS DFSMS Access Method Services for Catalogs*, SC26-7394.

6.11 Defining and deleting data sets



```
//DELDS   JOB    ...
//STEP1   EXEC PGM=IDCAMS
//SYSPRINT  DD    SYSOUT=A
//SYSIN     DD    *
     DELETE TEST1.A
/*
```

Figure 6-18 Deleting a data set

Define data sets

There are many ways to define a data set. Define means to create in VTOC and to catalog in a ICF catalog (BCS plus VVDS). Examples are using IDCAMS **DEFINE CLUSTER** to create a VSAM data set or using a JCL DD statement to define a non-VSAM data set. If you define a VSAM data set or SMS-managed non-VSAM data set, an entry in the BCS, in the VTOC, and in the VVDS is created.

Since z/OS V1R7, an attempt to define a page data set in a catalog not pointed to by the running master causes an IDCAMS message, instead of it being executed and causing later problems.

Delete data sets

Delete a data set implies uncataloging the data set, and *scratch* means take it out from VTOC. You can delete, for example, by running an IDCAMS **DELETE** job, by using a JCL DD statement, or by issuing the command **DELETE** in ISPF 3.4 in front of the data set name. The default of the **DELETE** command is scratch, which means the BCS, VTOC, and VVDS data set entries are erased. By doing that, the reserved space for this data set on the volume is released. The data set itself is not overwritten until the freed space is reused by another data set. You can use the parameter **ERASE** for an IDCAMS **DELETE** if you want the data set to be overwritten with binary zeros for security reasons.

Sometimes in your installation you need to delete an alias, delete only a catalog entry, or you may have orphaned catalog information on your DASD and need to delete a VSAM or a non-VSAM data set. The next few topics describe some of these situations and show you how to handle them.

Delete aliases

To simply delete an alias, use the IDCAMS **DELETE ALIAS** command, specifying the alias you are deleting. To delete all the aliases for a catalog, use **EXPORT DISCONNECT** to disconnect the catalog. The aliases are deleted when the catalog is disconnected. When you again connect the catalog (using **IMPORT CONNECT**) the aliases remain deleted.

Delete the catalog entry

To only delete a catalog entry you can use the **DELETE NOSCRATCH** command; the VVDS and VTOC entries are not deleted. The entry deleted can be reinstated with the **DEFINE RECATALOG** command as shown in Figure 6-19 on page 325.

```
//DELDS  JOB   ...
//STEP1  EXEC PGM=IDCAMS
//SYSPRINT  DD    SYSOUT=A
//SYSIN     DD    *
     DELETE TEST1.A NOSCRATCH   /* deletes TEST1.A from the BCS   */
     DEFINE NONVSAM       -     /* redefines TEST1.A into the BCS */
         (NAME(TEST1.A)   -
          DEVICETYPE(3390)-
          VOLUMES(VSF6S3) -
          RECATALOG       -
         )
/*
```

Figure 6-19 Delete NOSCRATCH, define RECATALOG

Delete VVR or NVR records

When the catalog entry is missing, and the data set remains on the DASD, you can use the **DELETE VVR** for VSAM data sets and **DELETE NVR** for non-VSAM SMS-managed data sets to remove its entry from the VVDS. You can only use these commands if there is no entry in the BCS for the data set.

```
//DELDS  JOB   ...
//S1      EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//DD1       DD VOL=SER=VSF6S3,UNIT=3390,DISP=OLD
//SYSIN     DD *
   DELETE TEST1.A -
          FILE(DD1) -
          NVR
/*
```

Figure 6-20 Delete the VVDS entry for a non-VSAM data set

Caution: When deleting VSAM KSDS, you must issue a **DELETE VVR** for each of the components, the DATA, and the INDEX.

Delete generation data groups

In this example, a generation data group (GDG) base catalog entry is deleted from the catalog. The generation data sets associated with GDGBASE remain unaffected in the VTOC, as shown in Figure 6-21.

```
//DELGDG JOB ...
//S1      EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
   DELETE TEST1.GDG           -
          GENERATIONDATAGROUP  -
          RECOVERY
/*
```

Figure 6-21 Delete a GDG base for recovery

The **DELETE** command with keyword **RECOVERY** removes the GDG base catalog entry from the catalog.

Delete an ICF

When deleting an ICF, you must take care to specify whether you want to delete only the catalog, or if you want to delete all associated data. The following examples show how to delete a catalog.

- Delete with recovery

  In Figure 6-22, a user catalog is deleted in preparation for replacing it with an imported backup copy. The VVDS and VTOC entries for objects defined in the catalog are not deleted and the data sets are not scratched, as shown in the JCL.

```
//DELET13    JOB   ...
//STEP1     EXEC  PGM=IDCAMS
//DD1       DD    VOL=SER=VSER01,UNIT=3390,DISP=OLD
//SYSPRINT  DD    SYSOUT=A
//SYSIN     DD    *
     DELETE -
            USER.CATALOG -
            FILE(DD1) -
            RECOVERY -
            USERCATALOG
/*
```

Figure 6-22 Delete a user catalog for recovery

  RECOVERY specifies that only the catalog data set is deleted, without deleting the objects defined in the catalog.

- Delete an empty user catalog

  In Figure 6-23 on page 327, a user catalog is deleted. A user catalog can be deleted when it is empty; that is, when there are no objects cataloged in it other than the catalog's volume. If the catalog is not empty, it cannot be deleted unless the **FORCE** parameter is specified.

```
//DELET6  JOB   ...
//STEP1    EXEC  PGM=IDCAMS
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
     DELETE -
           USER.CATALOG -
           PURGE        -
           USERCATALOG
/*
```

Figure 6-23 Delete an empty user catalog

Attention: The **FORCE** parameter deletes all data sets in the catalog. The **DELETE** command deletes both the catalog and the catalog's user catalog connector entry in the master catalog.

For more information about the **DELETE** command, refer to *z/OS DFSMS Access Method Services for Catalogs*, SC26-7394.

Delete a migrated data set

A *migrated* data set is a data set moved by DFSMShsm to a cheaper storage device to make room in your primary DASD farm. Catalog management recognizes that a data set is migrated by the MIGRAT volser in its catalog entry. A migrated data set can be **DELETE SCRATCH** (TSO **DELETE** issues the DFSMShsm **HDELETE** command) or **NOSCRATCH**.

Where:

SCRATCH The non-VSAM data set being deleted from the catalog is to be removed from the VTOC of the volume on which it resides. When SCRATCH is specified for a cluster, alternate index, page space, or data space, the VTOC entries for the volumes involved are updated to reflect the deletion of the object.

NOSCRATCH The non-VSAM data set being deleted from the catalog is to remain in the VTOC of the volume on which it resides or it has already been scratched from the VTOC. When NOSCRATCH is specified for a cluster, page space, alternate index, or data space, the VTOC entries for the volumes involved are not updated.

To execute the **DELETE** command against a migrated data set, you must have RACF group ARCCATGP defined. In general to allow certain authorized users to perform these operations on migrated data sets without recalling them, perform the following steps:

1. Define a RACF catalog maintenance group named ARCCATGP.

   ```
   ADDGROUP (ARCCATGP)
   ```

2. Connect the desired users to that group.

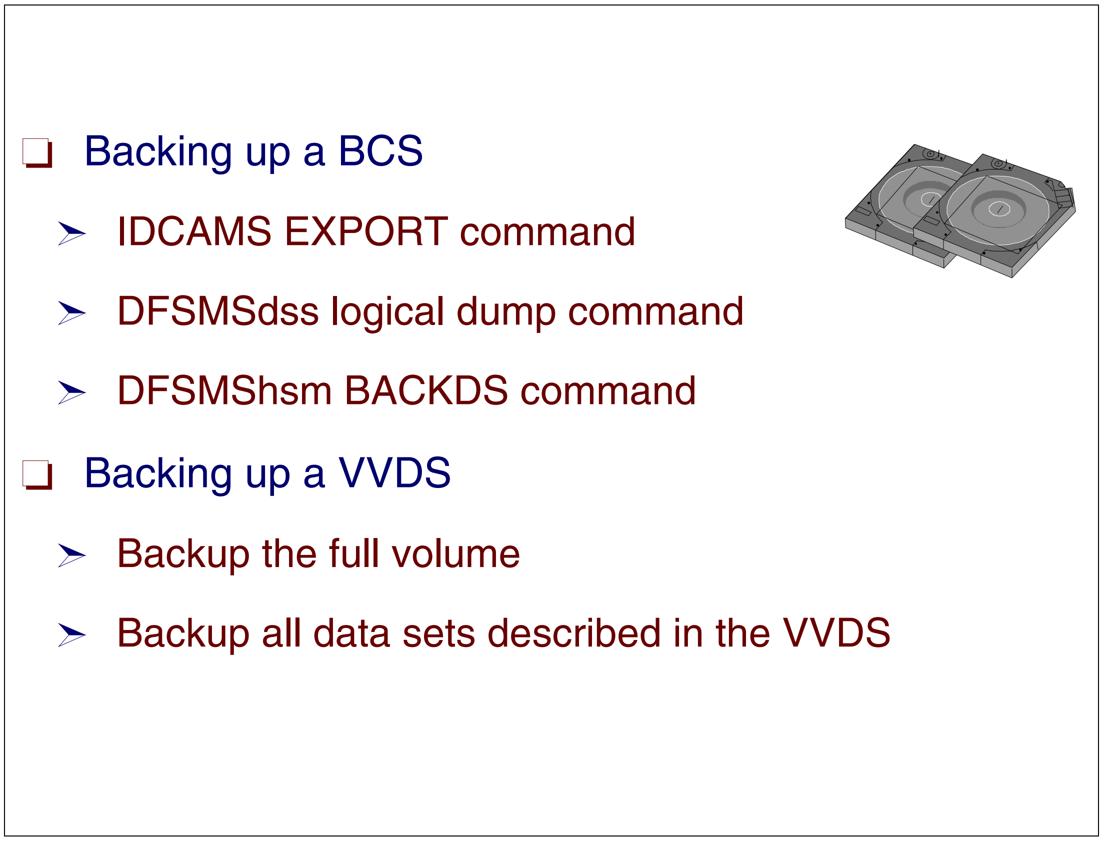
Only when such a user is logged on under group ARCCATGP does DFSMShsm bypass the automatic recall for UNCATALOG, RECATALOG, and DELETE/NOSCRATCH requests for migrated data sets. For example, the following LOGON command demonstrates starting a TSO session under ARCCATGP:

```
LOGON userid | password GROUP(ARCCATGP)
```

For further information about ARCCATGP group, refer to *z/OS DFSMShsm Implementation and Customization Guide*, SC35-0418.

When you need to delete a migrated data set, but the data set is not recorded in the HSM control data sets, you should execute a **DELETE NOSCRATCH** command for the data set to clean up the ICF catalog.

6.12 Backup procedures



- Backing up a BCS
  - IDCAMS EXPORT command
  - DFSMSdss logical dump command
  - DFSMShsm BACKDS command
- Backing up a VVDS
  - Backup the full volume
  - Backup all data sets described in the VVDS

Figure 6-24 Backup procedures for catalogs

Backup procedures

The two parts of an ICF catalog, the BCS and the VVDS, require different backup techniques. The BCS can be backed up like any other data set, whereas the VVDS should only be backed up as part of a volume dump. The entries in the VVDS and VTOC are backed up when the data sets they describe are:

- Exported with IDCAMS
- Logically dumped with DFSMSdss
- Backed up with DFSMShsm

Important: Because catalogs are essential system data sets, it is important that you maintain backup copies. The more recent and accurate a backup copy, the less impact a catalog outage will have on your installation.

Backing up a BCS

To back up a BCS you can use one of the following methods:

- The access method services **EXPORT** command
- The DFSMSdss logical **DUMP** command
- The DFSMShsm **BACKDS** command

You can later recover the backup copies using the same utility used to create the backup:

- The access method services **IMPORT** command for exported copies
- The DFSMSdss **RESTORE** command for logical dump copies
- The DFSMShsm **RECOVER** command for DFSMShsm backups

The copy created by these utilities is a *portable* sequential data set that can be stored on a tape or direct access device, which can be of a different device type than the one containing the source catalog.

When these commands are used to back up a BCS, the aliases of the catalog are saved in the backup copy. The source catalog is not deleted, and remains as a fully functional catalog. The relationships between the BCS and VVDSs are unchanged.

You cannot permanently export a catalog by using the PERMANENT parameter of **EXPORT**. The TEMPORARY option is used even if you specify PERMANENT or allow it to default. Figure 6-25 shows you an example for an IDCAMS EXPORT.

```
//EXPRTCAT JOB   ...
//STEP1    EXEC  PGM=IDCAMS
//RECEIVE  DD    DSNAME=CATBACK,UNIT=(TAPE,,DEFER),
//         DISP=(NEW,KEEP),VOL=SER=327409,LABEL=(1,SL)
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
     EXPORT -
           USER.CATALOG     -
           OUTFILE(RECEIVE) -
           TEMPORARY
/*
```

Figure 6-25 JCL to create a backup of a BCS using IDCAMS EXPORT

Note: You cannot use **IDCAMS REPRO** or other copying commands to create and recover BCS backups.

Backing up a master catalog

A master catalog can be backed up like any other BCS. You should use IDCAMS, DFSMSdss, or DFSMShsm for the backup. Another way to provide a backup for the master catalog is to create an alternate master catalog. For information about defining and using an alternate master catalog, see *z/OS DFSMS: Managing Catalogs*, SC26-7409.

You should also make periodic volume dumps of the master catalog's volume. This dump can later be used by the stand-alone version of DFSMSdss to restore the master catalog if you cannot access the volume from another system.

Backing up a VVDS

The VVDS should not be backed up as a data set to provide for recovery. To back up the VVDS, back up the volume containing the VVDS, or back up all data sets described in the VVDS (all VSAM and SMS-managed data sets). If the VVDS ever needs to be recovered, recover the entire volume, or all the data sets described in the VVDS.

You can use either DFSMSdss or DFSMShsm to back up and recover a volume or individual data sets on the volume.

6.13 Recovery procedures

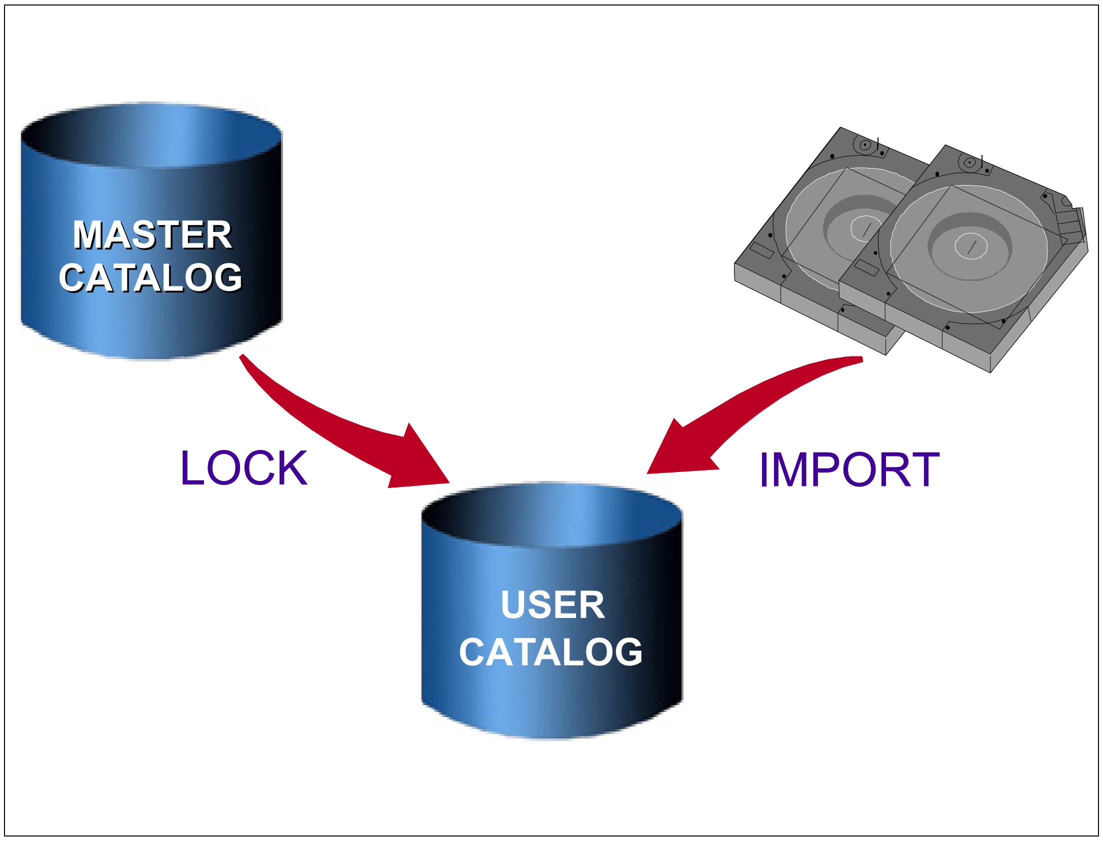


Figure 6-26 Recovery procedures

Recovery procedures

Before you run the recovery procedures mentioned in this section, you should also read 6.22, "Fixing temporary catalog problems" on page 349.

Normally, a BCS is recovered separately from a VVDS. A VVDS usually does not need to be recovered, even if an associated BCS is recovered. However, if you need to recover a VVDS, and a BCS resides on the VVDS's volume, you must recover the BCS as well. If possible, you should export the BCS before recovering the volume, and then recover the BCS from the exported copy. This ensures a current BCS.

Before recovering a BCS or VVDS, try to recover single damaged records. If damaged records can be rebuilt, you can avoid a full recovery.

Single BCS records can be recovered using the IDCAMS **DELETE** and **DEFINE** commands as described in 6.11, "Defining and deleting data sets" on page 324. Single VVDS and VTOC records can be recovered using the IDCAMS **DELETE** command and by recovering the data sets on the volume.

The way you recover a BCS depends on how it was saved (see 6.12, "Backup procedures" on page 329). When you recover a BCS, you do not need to delete and redefine the target catalog unless you want to change the catalog's size or other characteristics, or unless the BCS is damaged in such a way as to prevent the usual recovery.

Aliases to the catalog can be defined if you use DFSMSdss, DFSMShsm, or if you specify ALIAS on the **IMPORT** command. If you have not deleted and redefined the catalog, all existing aliases are maintained, and any aliases defined in the backup copy are redefined if they are not already defined.

Lock the BCS before you start recovery so that no one else has access to it while you recover the BCS. If you do not restrict access to the catalog, users might be able to update the catalog during recovery or maintenance and create a data integrity exposure. The catalog also will be unavailable to any system that shares the catalog. You cannot lock a master catalog.

After you recover the catalog, update the BCS with any changes which have occurred since the last backup, for example, by running IDCAMS **DEFINE RECATALOG** for all missing entries. You can use the access method services **DIAGNOSE** command to identify certain unsynchronized entries.

The Integrated Catalog Forward Recovery Utility

You also can use the *Integrated Catalog Forward Recovery Utility* (ICFRU) to recover a damaged catalog to a correct and current status. This utility uses SMF records that record changes to the catalog, updating the catalog with changes made since the BCS was backed up. The SMF records are used by ICFRU as a log in data base. We recommend the use of ICFRU to avoid the loss of some catalog data even after recovery.

Recovery step by step

Follow these steps to recover a BCS using the IDCAMS **IMPORT** command:

1. If the catalog is used by the job scheduler for any batch jobs, hold the job queue for all job classes except the one you use for the recovery.
2. Lock the catalog using the IDCAMS **ALTER LOCK** command.
3. Use ICFRU to create an *updated* version of your last EXPORT backup.
4. Import the most current backup copy of the BCS (which contains the BCS's aliases as they existed when the backup was made). For example, use this JCL:

```
//RECOVER  EXEC PGM=IDCAMS
//BACKCOPY DD DSN=BACKUP.SYS1.ICFCAT.PROJECT1,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
    IMPORT INFILE(BACKCOPY) -
         OUTDATASET(SYS1.ICFCAT.PROJECT1) -
         ALIAS -
         LOCK
```

5. If you did not run step 3, manually update the catalog with the changes made between the last backup and the time of error, for example by using IDCAMS **DEFINE RECATALOG**.
6. Use IDCAMS **DIAGNOSE** and **EXAMINE** commands to check the contents and integrity of the catalog (see 6.14, "Checking the integrity on an ICF structure" on page 333).
7. If the catalog is shared by other systems and was disconnect there for recovery, run IDCAMS **IMPORT CONNECT ALIAS** on those systems to reconnect the user catalog to the master catalog.
8. Unlock the catalog using IDCAMS **ALTER UNLOCK**.
9. Free the job queue if you put it on hold.

For further information about recovery procedures, see *z/OS DFSMS: Managing Catalogs*, SC26-7409. For information about the IDCAMS facility, see *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

6.14 Checking the integrity on an ICF structure

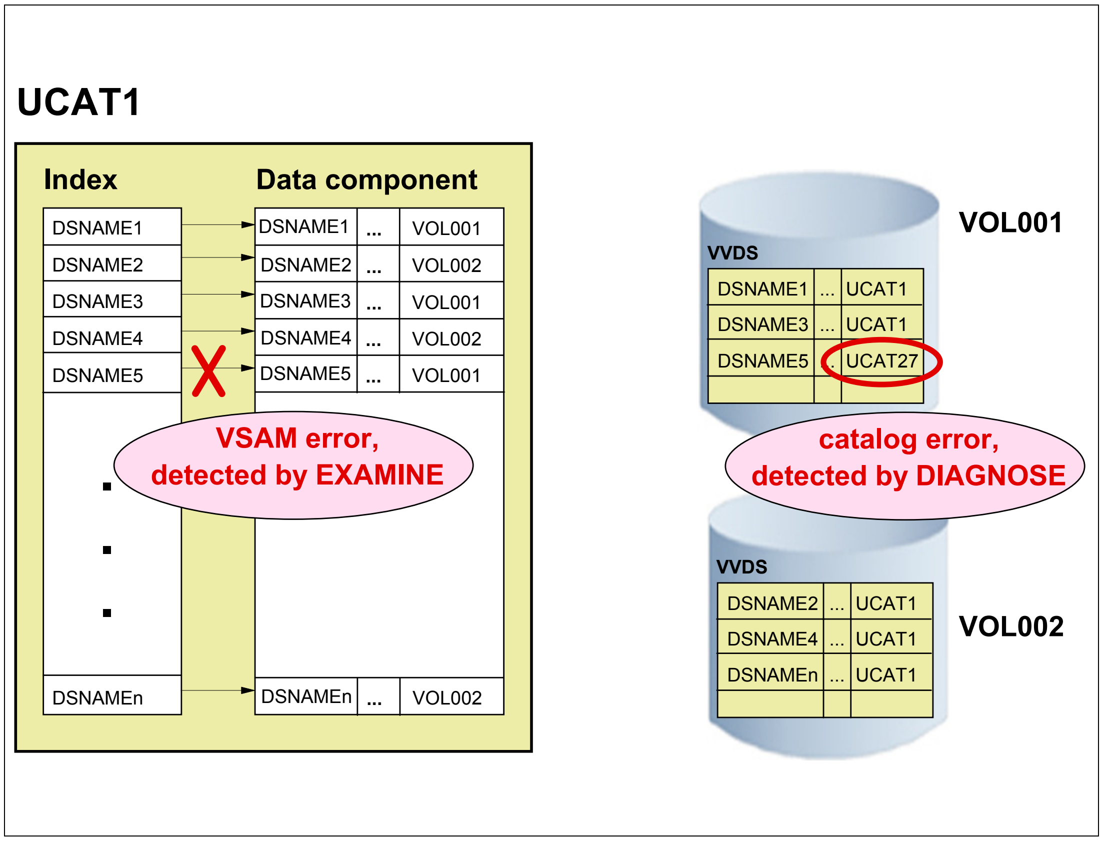


Figure 6-27 Errors in an ICF structure

Two types of errors

We need to differentiate between two types of errors in the ICF structure that cause the need for recovery. They are:

- Errors in the structural integrity of the BCS or VVDS as VSAM data sets - *VSAM error*

  Errors in the structure of a BCS as a VSAM KSDS or the VVDS as a VSAM ESDS data set usually mean that the data set is broken (logically or physically). The data set no longer has a valid structure that the VSAM component can handle. VSAM does not care about the contents of the records in the BCS or VVDS.

- Errors within the data structure of a BCS or VVDS - *catalog error*

  The VSAM structure of the BCS or VVDS is still valid. VSAM has no problems accessing the data set. However, the content of the single records in the BCS or VVDS does not conform with the catalog standards. The information in the BCS and VVDS for a single data set could be unsynchronized, making the data set inaccessible.

VSAM errors

There are two kinds of VSAM errors that could happen to your BCS or VVDS:

- Logical errors

  The records on the DASD volume still have valid physical characteristics like record size or CI size. The VSAM information in those records is wrong, like pointers from one record to another or the end-of-file information.

- Physical errors

  The records on the DASD volume are invalid; for example, they are of a wrong length. Reasons can be an overlay of physical DASD space or wrong extent information for the data set in the VTOC or VVDS.

When errors in the VSAM structure occur, they are in most cases logical errors for the BCS. Because the VVDS is an *entry-sequenced* data set (ESDS), it has no index component. Logical errors for an ESDS are unlikely.

You can use the IDCAMS **EXAMINE** command to analyze the structure of the BCS. As explained previously, the BCS is a VSAM *key-sequenced* data set (KSDS). Before running the **EXAMINE**, you should run an IDCAMS **VERIFY** to make sure that the VSAM information is current, and **ALTER LOCK** the catalog to prevent update from others while you are inspecting it.

With the parameter **INDEXTEST** you analyze the integrity of the index. With parameter **DATATEST** you analyze the data component. If only the index test shows errors, you might have the chance to recover the BCS by just running an **EXPORT**/**IMPORT** to rebuild the index. If there is an error in the data component, you probably have to recover the BCS as described in “Recovery procedures” on page 331.

Catalog errors

By *catalog errors* we mean errors in the catalog information of a BCS or VVDS, or unsynchronized information between the BCS and VVDS. The VSAM structure of the BCS is still valid, that is, an **EXAMINE** returns no errors.

Catalog errors can make a data set inaccessible. Sometimes it is sufficient to delete the affected entries, sometimes the catalog needs to be recovered (see “Recovery procedures” on page 331).

You can use the IDCAMS **DIAGNOSE** command to validate the contents of a BCS or VVDS. You can use this command to check a single BCS or VVDS and to compare the information between a BCS and multiple VVDSs.

For various **DIAGNOSE** examples see *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

6.15 Protecting catalogs

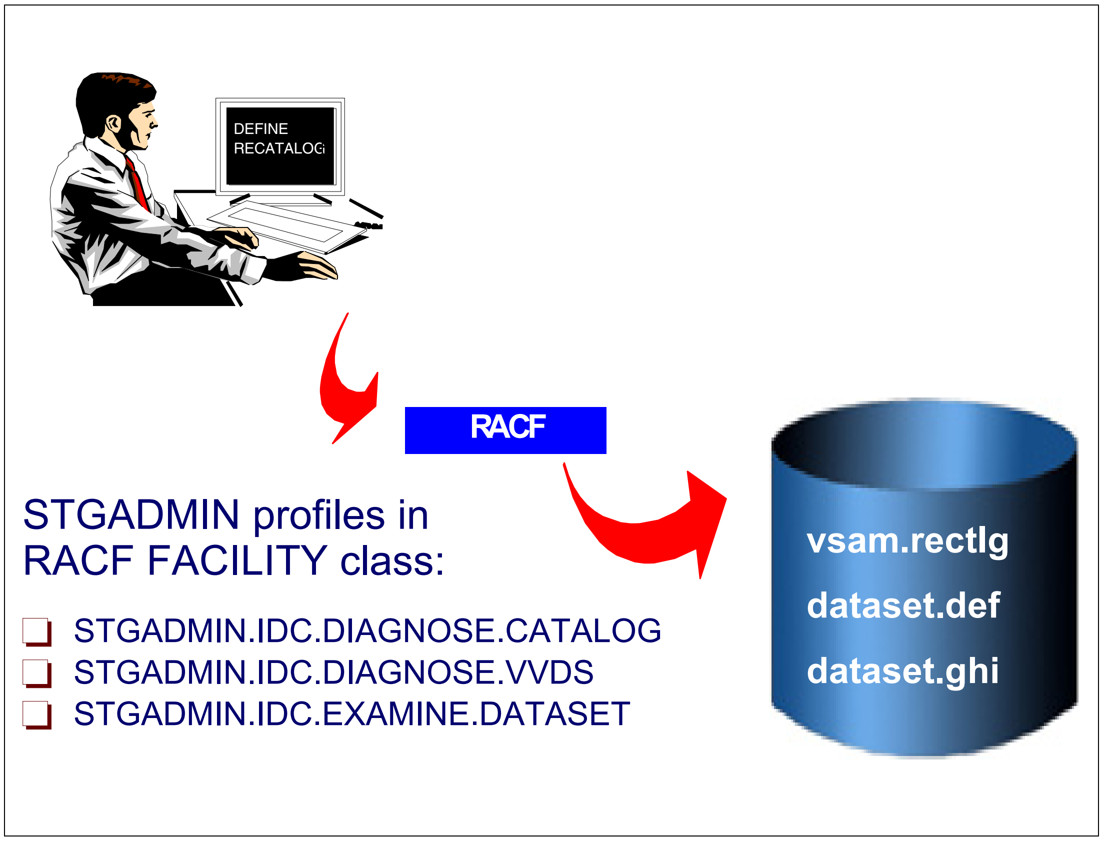

DEFINE
RECATALOG
RACF
STGADMIN profiles in RACF FACILITY class:
STGADMIN.IDC.DIAGNOSE.CATALOG
STGADMIN.IDC.DIAGNOSE.VVDS
STGADMIN.IDC.EXAMINE.DATASET
vsam.rectlg
dataset.def
dataset.ghi

Figure 6-28 Protecting catalogs with RACF

Protecting catalogs

The protection of data includes:

- Data security: the safety of data from theft or intentional destruction
- Data integrity: the safety of data from accidental loss or destruction

Data can be protected either indirectly, by preventing access to programs that can be used to modify data, or directly, by preventing access to the data itself. Catalogs and cataloged data sets can be protected in both ways.

To protect your catalogs and cataloged data, use the Resource Access Control Facility (RACF) or a similar product.

Authorized program facility (APF) protection

The authorized program facility (APF) limits the use of sensitive system services and resources to *authorized* system and user programs.

For information about using APF for program authorization, see *z/OS MVS Programming: Authorized Assembler Services Guide,* SA22-7608.

All IDCAMS load modules are contained in SYS1.LINKLIB, and the root segment load module (IDCAMS) is link-edited with the SETCODE AC(1) attribute. These two characteristics ensure that access method services executes with APF authorization.

Because APF authorization is established at the job step task level, access method services is not authorized if invoked by an unauthorized application or terminal monitor program.

RACF authorization checking

RACF provides a software access control measure you can use in addition to or instead of passwords. RACF protection and password protection can coexist for the same data set.

To open a catalog as a data set, you must have ALTER authority and APF authorization. When defining an SMS-managed data set, the system only checks to make sure the user has authority to the data set name and SMS classes and groups. The system selects the appropriate catalog, without checking the user's authority to the catalog. You can define a data set if you have ALTER or OPERATIONS authority to the applicable data set profile.

Deleting any type of RACF-protected entry from a RACF-protected catalog requires ALTER authorization to the catalog or to the data set profile protecting the entry being deleted. If a non-VSAM data set is SMS-managed, RACF does not check for DASDVOL authority. If a non-VSAM, non-SMS-managed data set is being scratched, DASDVOL authority is also checked.

For ALTER RENAME, the user is required to have the following two types of authority:

- ALTER authority to either the data set or the catalog
- ALTER authority to the new name (generic profile) or CREATE authority to the group

Be sure that RACF profiles are correct after you use REPRO MERGECAT or CNVTCAT on a catalog that uses RACF profiles. If the target and source catalogs are on the same volume, the RACF profiles remain unchanged.

Tape data sets defined in an integrated catalog facility catalog can be protected by:

- Controlling access to the tape volumes
- Controlling access to the individual data sets on the tape volumes

Profiles

To control the ability to perform functions associated with storage management, define profiles in the FACILITY class whose profile names begin with STGADMIN (storage administration). For a complete list of STGADMIN profiles, see *z/OS DFSMSdfp Storage Administration Reference*, SC26-7402. Examples of some profiles are:

```
STGADMIN.IDC.DIAGNOSE.CATALOG
STGADMIN.IDC.DIAGNOSE.VVDS
STGADMIN.IDC.EXAMINE.DATASET
```

6.16 Merging catalogs

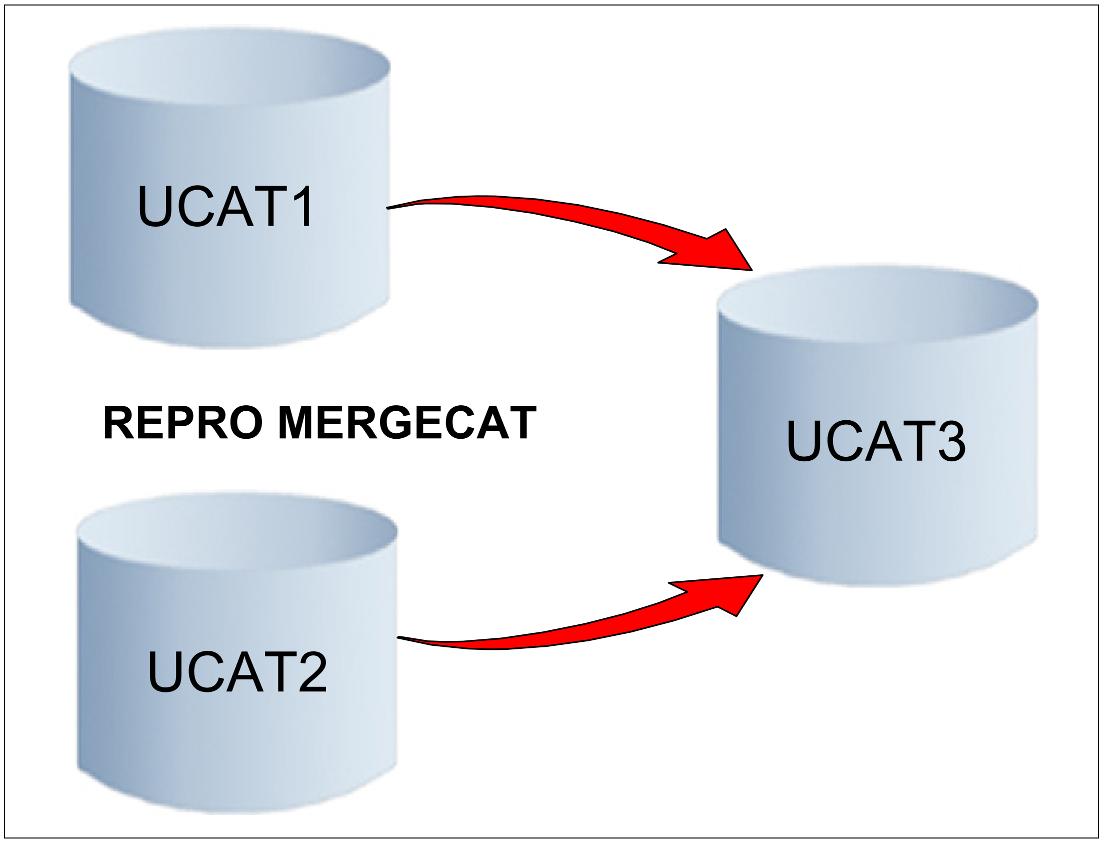


Figure 6-29 Merging catalogs

Merging catalogs

You might find it beneficial to merge catalogs if you have many small or seldom-used catalogs. An excessive number of catalogs can complicate recovery procedures and waste resources such as CAS storage, tape mounts for backups, and system time performing backups.

Merging catalogs is accomplished in much the same way as splitting catalogs (see “Splitting a catalog” on page 339). The only difference between splitting catalogs and merging them is that in merging, you want all the entries in a catalog to be moved to a different catalog, so that you can delete the obsolete catalog.

Use the following steps to merge two integrated catalog facility catalogs:

1. Use **ALTER LOCK** to lock both catalogs.
2. Use **LISTCAT** to list the aliases for the catalog you intend to delete after the merger:

```
//JOB    ...
//S1      EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//DD1     DD   DSN=listcat.output,DISP=(NEW,CATLG),
//             SPACE=(TRK,(10,10)),
//             DCB=(RECFM=VBA,LRECL=125,BLKSIZE=629)
//SYSIN     DD *
  LISTC ENT(catalog.name) ALL -
    OUTFILE(DD1)
/*
```

3. Use **EXAMINE** and **DIAGNOSE** to ensure that the catalogs are error-free. Fix any errors indicated (see also "Checking the integrity on an ICF structure" on page 333).
4. Use **REPRO MERGECAT** *without* specifying the ENTRIES or LEVEL parameter. The OUTDATASET parameter specifies the catalog that you are keeping after the two catalogs are merged. Here is an example:

```
//MERGE6   JOB   ...
//STEP1    EXEC  PGM=IDCAMS
//DD1      DD    VOL=SER=VSER01,UNIT=DISK,DISP=OLD
//         DD    VOL=SER=VSER02,UNIT=DISK,DISP=OLD
//         DD    VOL=SER=VSER03,UNIT=DISK,DISP=OLD
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
     REPRO -
           INDATASET(USERCAT4) -
           OUTDATASET(USERCAT5) -
           MERGECAT -
           FILE(DD1)
            FROMKEY(WELCHA.C.*) TOKEY(WELCHA.E.*)
/*
```

Important: This step can take a long time to complete. If the MERGECAT job is cancelled for some reason, all merged entries so far remain in the *target* catalog. They are not backed out in case the job fails. See also "Recovering from a REPRO MERGECAT Failure" in *z/OS DFSMS: Managing Catalogs*, SC26-7409.

Since z/OS V1R7, **REPRO MERGECAT** provides the capability to copy a range of records from one user catalog to another. It allows recovery of a broken catalog by enabling you to copy from one specific key to another specific key just before where the break occurred and then recover data beginning after the break. Refer to the parameters **FROMKEY/TOKEY** in the previous example.

5. Use the listing created in step 2 to create a sequence of **DELETE ALIAS** and **DEFINE ALIAS** commands to delete the aliases of the obsolete catalog, and to redefine the aliases as aliases of the catalog you are keeping.

   The **DELETE ALIAS/DEFINE ALIAS** sequence must be run on each system that shares the changed catalogs and uses a different master catalog.
6. Use **DELETE USERCATALOG** to delete the obsolete catalog. Specify RECOVERY on the **DELETE** command.
7. If your catalog is shared, run the **EXPORT DISCONNECT** command on each shared system to remove unwanted user catalog connector entries. If your catalog is shared, run the **EXPORT DISCONNECT** command on each shared system to remove unwanted user catalog connector entries.
8. Use **ALTER UNLOCK** to unlock the remaining catalog.

You can also merge entries from one tape volume catalog to another using **REPRO MERGECAT**. **REPRO** retrieves tape library or tape volume entries and redefines them in a target tape volume catalog. In this case, VOLUMEENTRIES needs to be used to correctly filter the appropriate entries. The LEVEL parameter is not allowed when merging tape volume catalogs.

6.17 Splitting a catalog

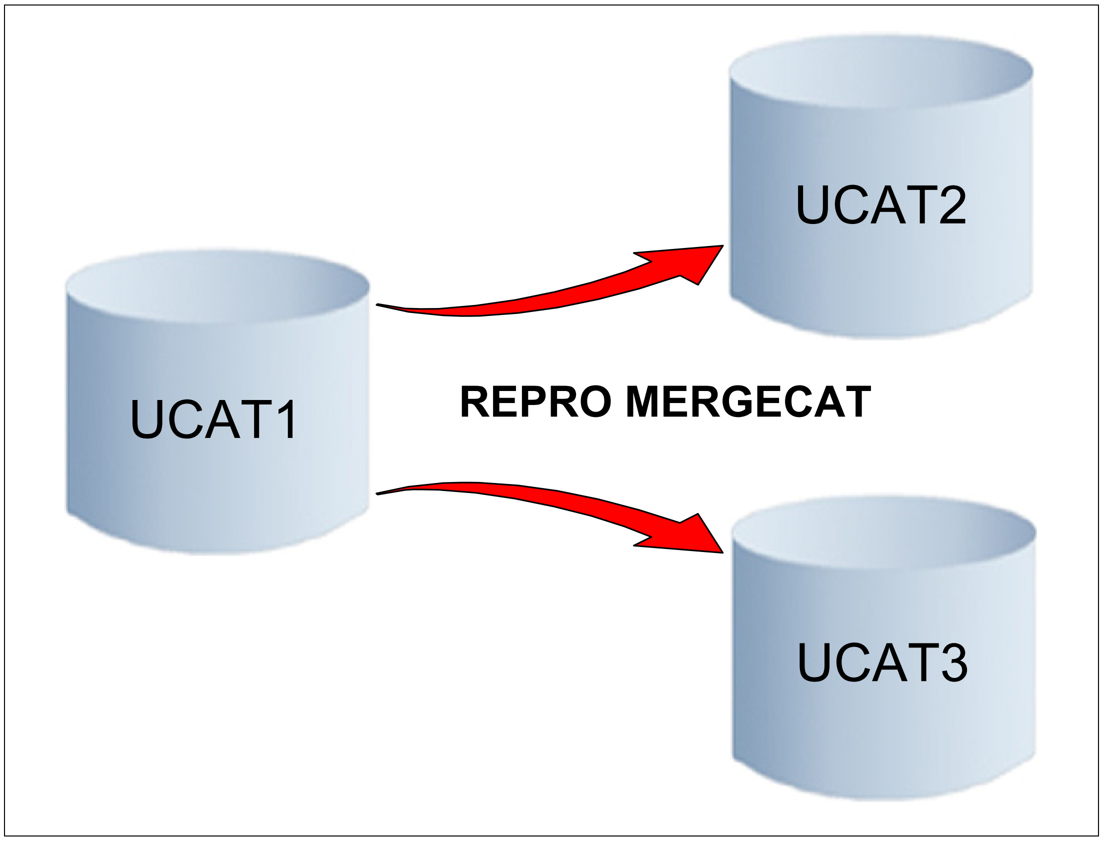


Figure 6-30 Splitting a catalog

Splitting catalogs

You can split a catalog to create two catalogs or to move a group of catalog entries if you determine that a catalog is either unacceptably large or that it contains too many entries for critical data sets.

If the catalog is unacceptably large (a catalog failure would leave too many entries inaccessible), then you can split the catalog into two catalogs. If the catalog is of an acceptable size but contains entries for too many critical data sets, then you can simply move entries from one catalog to another.

To split a catalog or move a group of entries, use the access method services **`REPRO MERGECAT`** command. Use the following steps to split a catalog or to move a group of entries:

1. Use **`ALTER LOCK`** to lock the catalog. If you are moving entries to an existing catalog, lock it as well.
2. If you are splitting a catalog, define a new catalog with **`DEFINE USERCATALOG LOCK`** (see also “Defining a catalog and its aliases” on page 316).
3. Use **`LISTCAT`** to obtain a listing of the catalog aliases you are moving to the new catalog. Use the OUTFILE parameter to define a data set to contain the output listing (see also “Merging catalogs” on page 337).
4. Use **`EXAMINE`** and **`DIAGNOSE`** to ensure that the catalogs are error-free. Fix any errors indicated (see also “Checking the integrity on an ICF structure” on page 333).

5. Use **REPRO MERGECAT** to split the catalog or move the group of entries. When splitting a catalog, the OUTDATASET parameter specifies the catalog created in step 2. When moving a group of entries, the OUTDATASET parameter specifies the catalog which is to receive the entries.

   Use the ENTRIES or LEVEL parameters to specify which catalog entries are to be removed from the source catalog and placed in the catalog specified in OUTDATASET.

   In the following example all entries that match the generic name VSAMDATA.* are moved from catalog USERCAT4 to USERCAT5.

```
//MERGE76   JOB   ...
//STEP1    EXEC  PGM=IDCAMS
//DD1      DD    VOL=SER=VSER01,UNIT=DISK,DISP=OLD
//         DD    VOL=SER=VSER02,UNIT=DISK,DISP=OLD
//         DD    VOL=SER=VSER03,UNIT=DISK,DISP=OLD
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
     REPRO -
           INDATASET(USERCAT4) -
           OUTDATASET(USERCAT5) -
           ENTRIES(VSAMDATA.*) -
           MERGECAT -
           FILE(DD1)
/*
```

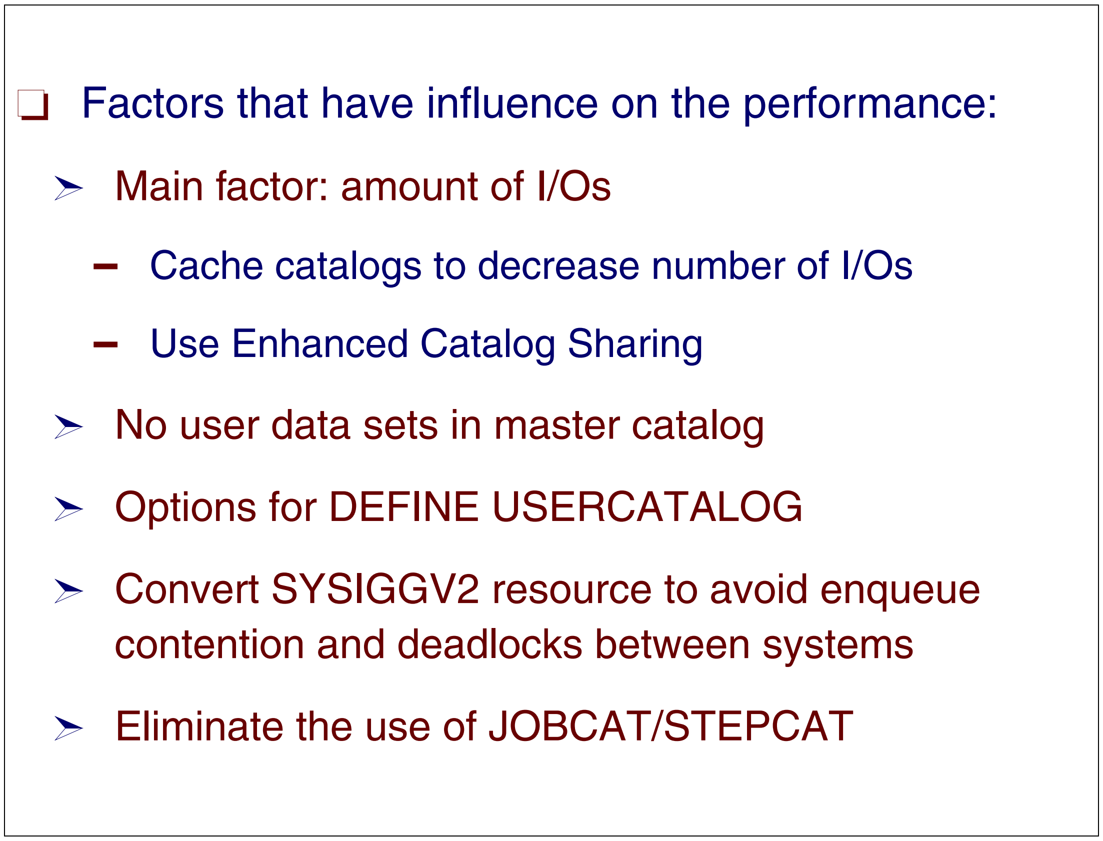
   Important: This step can take a long time to complete. If the MERGECAT job is cancelled for some reason, all merged entries so far will remain in the *target* catalog. They are not backed out in case the job fails. See also "Recovering from a REPRO MERGECAT Failure" in *z/OS DFSMS: Managing Catalogs*, SC26-7409.

6. Use the listing created in step 3 to create a sequence of **DELETE ALIAS** and **DEFINE ALIAS** commands for each alias. These commands delete the alias from the original catalog, and redefine them as aliases for the catalog which now contains entries belonging to that alias name.

   The **DELETE ALIAS/DEFINE ALIAS** sequence must be run on each system that shares the changed catalogs and uses a different master catalog.

7. Unlock both catalogs using **ALTER UNLOCK**.

6.18 Catalog performance



- Factors that have influence on the performance:
  - Main factor: amount of I/Os
    - Cache catalogs to decrease number of I/Os
    - Use Enhanced Catalog Sharing
  - No user data sets in master catalog
  - Options for DEFINE USERCATALOG
  - Convert SYSIGGV2 resource to avoid enqueue contention and deadlocks between systems
  - Eliminate the use of JOBCAT/STEPCAT

Figure 6-31 Catalog performance

Catalog performance

Performance should not be your main consideration when you define catalogs. It is more important to create a catalog configuration that allows easy recovery of damaged catalogs with the least amount of system disruption. However, there are several options you can choose to improve catalog performance without affecting the recoverability of a catalog. Remember that in an online environment, such as CICS/DB2, the number of data set allocations is minimal and consequently the catalog activity is low.

Factors affecting catalog performance

The main factors affecting catalog performance are the amount of I/O required for the catalog and the subsequent amount of time it takes to perform the I/O. These factors can be reduced by buffering catalogs in special buffer pools used only by CI catalogs. Other factors are the size and usage of the master catalog, options that were used to define a catalog, and the way you share catalogs between systems.

Buffering catalogs

The simplest method of improving catalog performance is to use a buffer to maintain catalog records within CAS private area address space or VLF data space.

Two types of buffer are available exclusively for catalogs. The in-storage catalog (ISC) buffer is contained within the catalog address space (CAS). The catalog data space buffer (CDSC) is separate from CAS and uses the z/OS VLF component, which stores the buffered records in a data space. Both types of buffer are optional, and each can be cancelled and restarted without an IPL.

The two types of buffer are used to keep catalog records in the storage. This avoids I/Os that would be necessary to read the records from DASD.

There are several things you need to take into considerations to decide what kind of buffer to use for which catalog. See *z/OS DFSMS: Managing Catalogs,* SC26-7409 for more information about buffering.

Another kind of caching is using Enhanced Catalog Sharing to avoid I/Os to read the catalog VVR. Refer to “Enhanced catalog sharing” on page 351 for information.

Master catalog

If the master catalog only contains entries for catalogs, catalog aliases, and system data sets, the *entire* master catalog is read into main storage during system initialization. Because the master catalog, if properly used, is rarely updated, the performance of the master catalog is not appreciably affected by I/O requirements. For that reason, keep the master catalog small and do not define user data sets into it.

Options for defining a user catalog

There are several options you can specify when you define a user catalog that have an impact on the performance. The options are:

- STRNO - Specifies the numbers of concurrent requests.
- BUFFERSPACE - Required buffer space for your catalog. It is determined by catalog management, but you can change it.
- BUFND - Number of buffers for transmitting data between virtual and DASD. The default is STRNO+1, but you can change it.
- BUFNI - Number of buffers for transmitting index entries between virtual and auxiliary storage. Default is STRNO+2.
- FREESPACE - an adequate value allows catalog updates without an excessive number of control interval and control area splits

For more information about these values see *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

Since z/OS V1R7, a new catalog auto-tuning (every 10-minutes) automatically modifies temporarily the number of data buffers, index buffers, and VSAM strings for catalogs. When any modification occurs the message `IEC391I` is issued telling the new values. This function is by default enabled, but can be disabled through the **`F CATALOG,DISABLE(AUTOTUNING)`**.

Convert SYSIGGV2 resource

Catalog management uses the SYSIGGV2 reserve while serializing access to catalogs. The SYSIGGV2 reserve is used to serialize the entire catalog BCS component across all I/O as well as to serialize access to specific catalog entries.

If the catalog is shared only within one GRSplex, you should convert the SYSIGGV2 resource to a global enqueue to avoid reserves on the volume on which the catalog resides. If you are not converting SYSIGGV2, you can have ENQ contentions on those volumes and you can even run into deadlock situations.

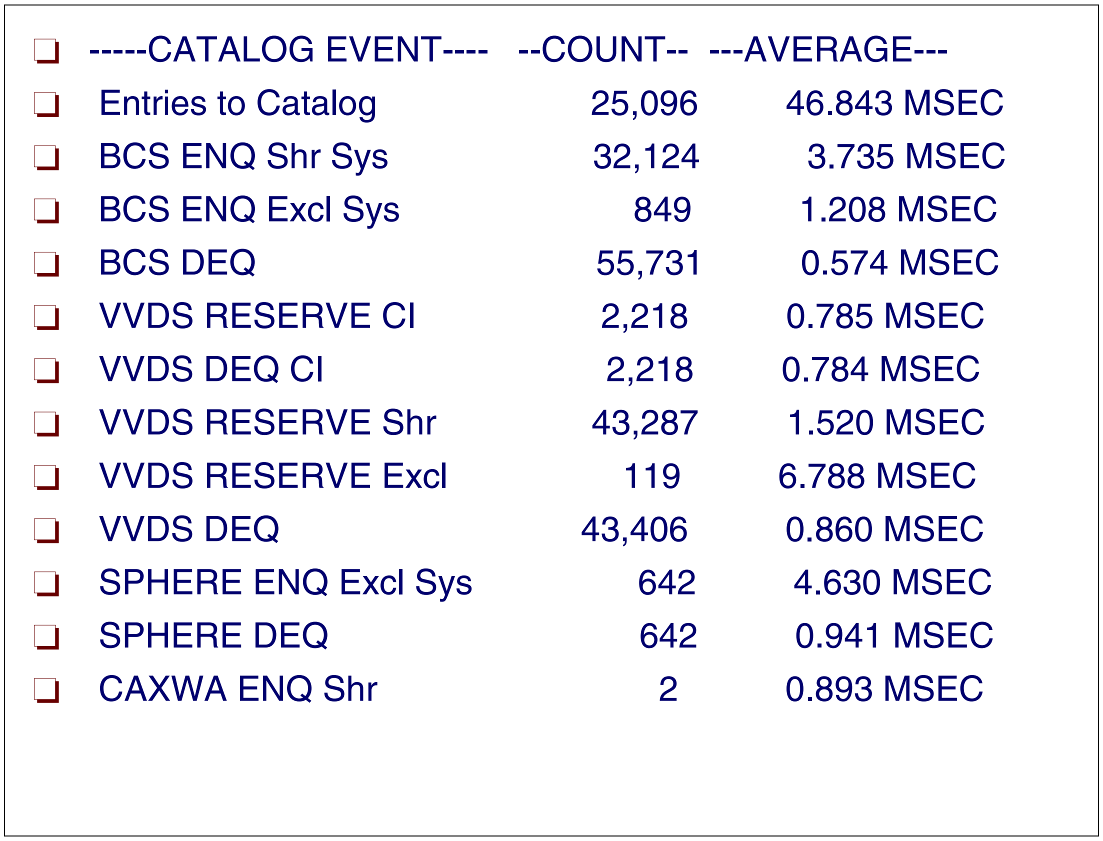
Attention: If you share a catalog with a system which is not in the same GRS complex, do not convert the SYSIGGV2 resource for this catalog. Sharing a catalog outside the complex requires reserves for the volume on which the catalog resides. Otherwise, you will break the catalog.

For more information refer to *z/OS MVS Planning: Global Resource Serialization,* SA22-7600.

Eliminate use of JOBCAT/STEPCAT

Eliminating JOBCAT and STEPCAT DD statements from job streams improves job and catalog performance, especially if two or more catalogs are concatenated on the DD statement. Catalog searches should be performed using catalog aliases. To prevent potential performance problems, eliminate all JOBCAT and STEPCAT DD statements from the job streams at your installation, and have every catalog request satisfied by catalog aliases. The default for the JOBCAT and STEPCAT option is to be disabled.

6.19 F CATALOG,REPORT,PERFORMANCE command



```
❑ -----CATALOG EVENT----   --COUNT--  ---AVERAGE---
❑  Entries to Catalog          25,096      46.843 MSEC
❑  BCS ENQ Shr Sys             32,124       3.735 MSEC
❑  BCS ENQ Excl Sys               849       1.208 MSEC
❑  BCS DEQ                     55,731       0.574 MSEC
❑  VVDS RESERVE CI              2,218       0.785 MSEC
❑  VVDS DEQ CI                  2,218       0.784 MSEC
❑  VVDS RESERVE Shr            43,287       1.520 MSEC
❑  VVDS RESERVE Excl              119       6.788 MSEC
❑  VVDS DEQ                    43,406       0.860 MSEC
❑  SPHERE ENQ Excl Sys            642       4.630 MSEC
❑  SPHERE DEQ                     642       0.941 MSEC
❑  CAXWA ENQ Shr                    2       0.893 MSEC
```

Figure 6-32 F CATALOG,REPORT,PERFORMANCE command

F CATALOG,REPORT,PERFORMANCE command

This `MODIFY` command is very important for performance analysis. Try to get familiar with each meaning in order to understand what can be done to improve the catalog performance. These counters can be zeroed through the use of a reset command.

Also the command F CATALOG,REPORT,CACHE produces rich information about the use of catalog buffering.

6.20 Catalog address space (CAS)

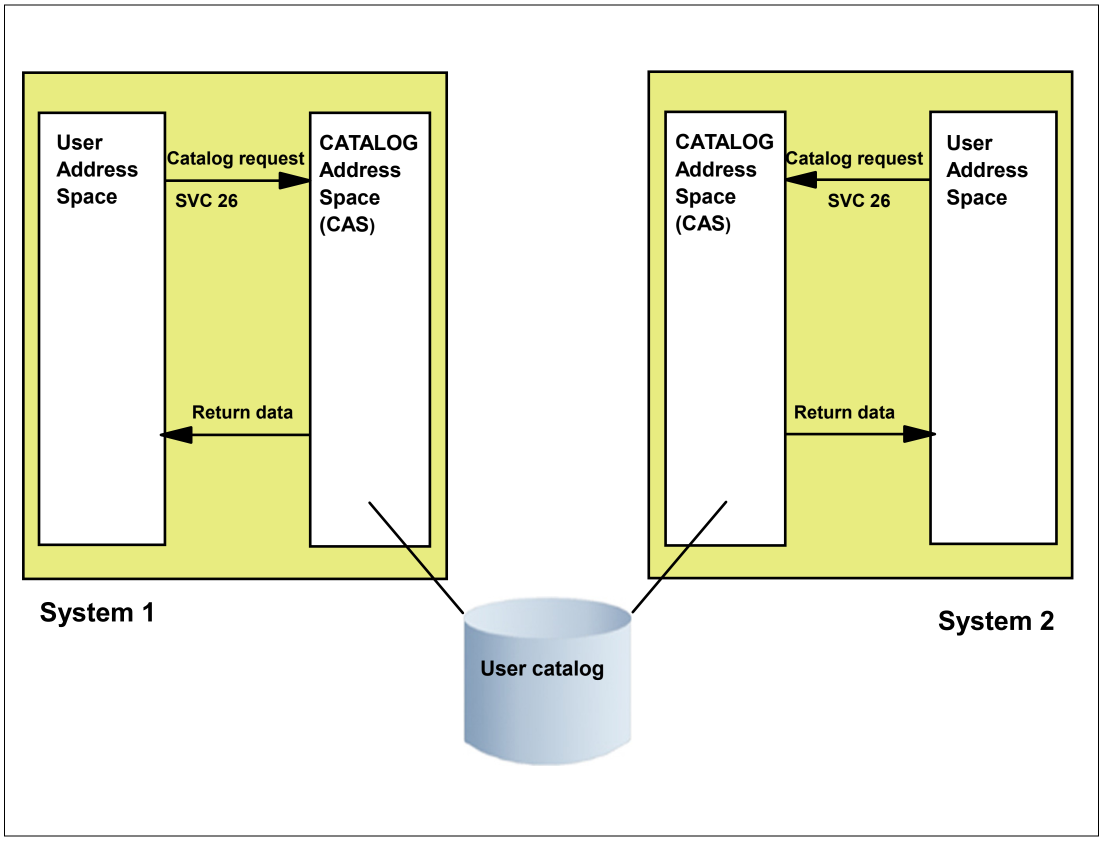


Figure 6-33 Catalog address space (CAS)

The catalog address space

Catalog functions are performed in the *catalog address space* (CAS). The jobname of the catalog address space is CATALOG.

As soon as a user requests a catalog function (for example, to locate or define a data set), the CAS gets control to handle the request. When it has finished, it returns the requested data to the user. A catalog task which handles a single user request is called a *service task*. To each user request a service task is assigned. The minimum number of available service tasks is specified in the SYSCATxx member of SYS1.NUCLEUS (or the LOADxx member of SYS1.PARMLIB). A table called the CRT keeps track of these service tasks.

The CAS contains all information necessary to handle a catalog request, like control block information about all open catalogs, alias tables, and buffered BCS records.

During the initialization of an MVS system, all user catalog names identified in the master catalog, their aliases, and their associated volume serial numbers are placed in tables in CAS.

You can use the **`MODIFY CATALOG`** operator command to work with the catalog address space. See also “Working with the catalog address space” on page 347.

Since z/OS 1.8 the maximum number of parallel catalog requests is 999, as defined in the SYSCAT parmlib member. Previously it was 180.

Restarting the catalog address space

Restarting the CAS should be considered only as a final option before IPLing a system. Never try restarting CAS unless an IPL is your only other option. A system failure caused by catalogs, or a CAS storage shortage due to FREEMAIN failures, might require you to use **`MODIFY CATALOG,RESTART`** to restart CAS in a new address space.

Never use **`RESTART`** to refresh catalog or VVDS control blocks or to change catalog characteristics. Restarting CAS is a drastic procedure, and if CAS cannot restart, you will have to IPL the system.

When you issue **`MODIFY CATALOG,RESTART`**, the CAS mother task is abended with abend code 81A, and any catalog requests in process at the time are redriven.

The restart of CAS in a new address space should be transparent to all users. However, even when all requests are redriven successfully and receive a return code of zero, the system might produce indicative dumps. There is no way to suppress these indicative dumps.

Since z/OS 1.6 the **`F CATALOG`** has new options:

`TAKEDUMP` This option causes the CAS to issue an SVCDUMP using the proper options to ensure that all data needed for diagnosis is available.

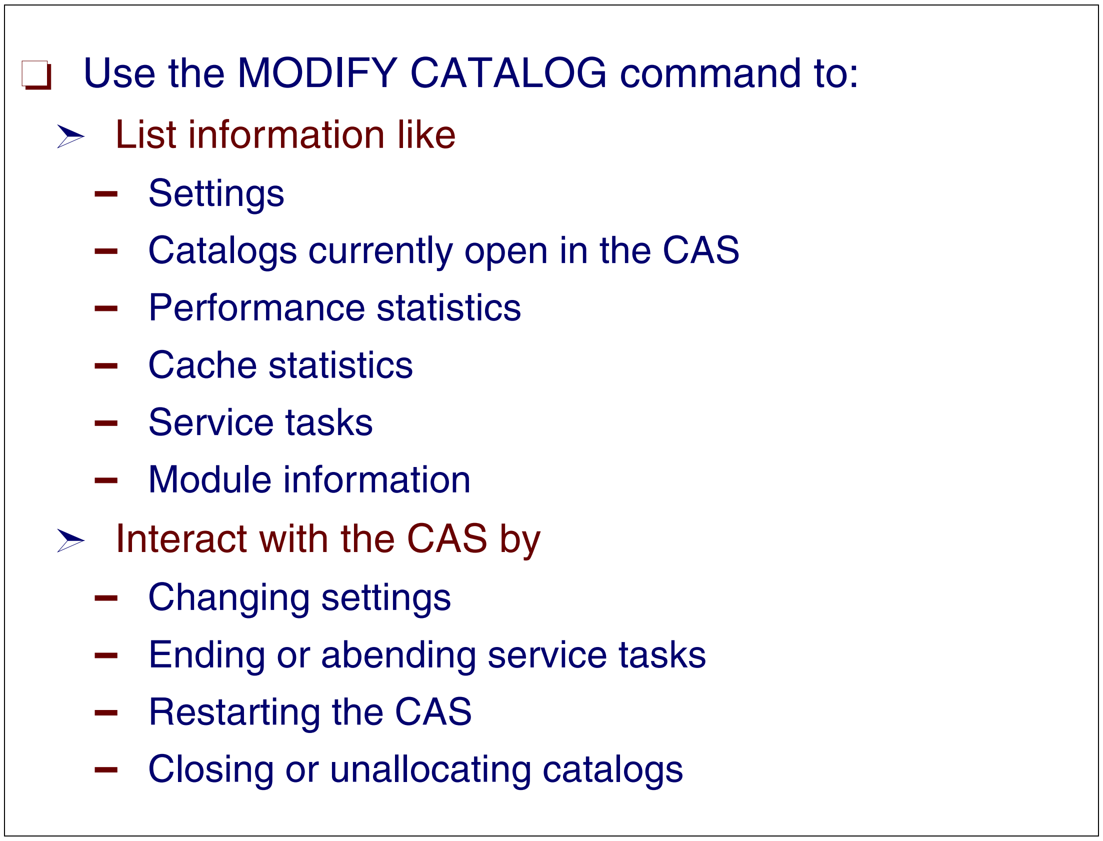
`RESTART` This option prompts the operator for additional information with the following messages:

- `IEC363D IS THIS RESTART RELATED TO AN EXISTING CATALOG PROBLEM (Y OR N)?`

  If the response to message IEC363D is `N`, the restart continues; if the response is `Y`, another prompt is issued.

- `IEC364D HAS AN SVC DUMP OF THE CATALOG ADDRESS SPACE ALREADY BEEN TAKEN (Y OR N)?`

6.21 Working with the catalog address space



- ❑ Use the MODIFY CATALOG command to:
  - ➢ List information like
    - – Settings
    - – Catalogs currently open in the CAS
    - – Performance statistics
    - – Cache statistics
    - – Service tasks
    - – Module information
  - ➢ Interact with the CAS by
    - – Changing settings
    - – Ending or abending service tasks
    - – Restarting the CAS
    - – Closing or unallocating catalogs

Figure 6-34 Working with the CAS

Working with the catalog address space

You can use the command **MODIFY CATALOG** to extract information from the CAS and to interact with the CAS. This command can be used in many variations. In this section we provide an overview of some parameters you should know to maintain your catalog environment. This command is further discussed in “Fixing temporary catalog problems” on page 349.

For a discussion about the entire functionality of the **MODIFY CATALOG** command, refer to *z/OS DFSMS: Managing Catalogs,* SC26-7409.

Examples of the **MODIFY CATALOG** command:

- **MODIFY CATALOG,REPORT**

  The catalog report lists information about the CAS, like the service level, the catalog address space ID, service tasks limits, and more. Since z/OS 1.7 this command generates the IEC392I that identifies the top three holders of the CATALOG service tasks, maybe indicating a lockup.

- **MODIFY CATALOG,OPEN**

  This command lists all catalogs that are currently open in the CAS. It shows whether the catalog is locked or shared, and the type of buffer used for the catalog. A catalog is opened after an IPL or catalog restart when it is referenced for the first time. It remains open until it is manually closed by the **MODIFY CATALOG** command or it is closed because the maximum number of open catalogs has been reached.

- **`MODIFY CATALOG,LIST`**

  The **`LIST`** command shows you all service task that are currently active handling a user request. Normally the active phase of a service task is only of a short duration, so no tasks are listed. This command helps you to identify tasks which could be the reason for catalog problems if the performance slows down.

- **`MODIFY CATALOG,DUMPON(rc,rsn,mm,cnt)`**

  Use this command to get a dump of the catalog address space when a specific catalog error occurs. This catalog error is identified by the catalog return code (rc), the reason code (rsn) and the name of the module which sets the error (mm). You can also specify, in the cnt-parameter, for how many of the rc/rsn-combinations a dump is to be taken. The default is one. The module identifier corresponds to the last two characters in the catalog module name. For example, the module identifier is A3 for IGG0CLA3. You can substitute the module name by two asterisks (**) if you don't know the module name or if you don't care about it.

- **`MODIFY CATALOG,ENTRY(modulename)`**

  This command lists the storage address, fmid, and the level of a catalog module (csect). If you do not specify a module name, all catalog csects are listed.

- **`MODIFY CATALOG,REPORT,PERFORMANCE`**

  The output of this command shows you significant catalog performance numbers about number and duration of ENQs and DEQs for the BCS and VVDS and more. Use the command to identify performance problems.

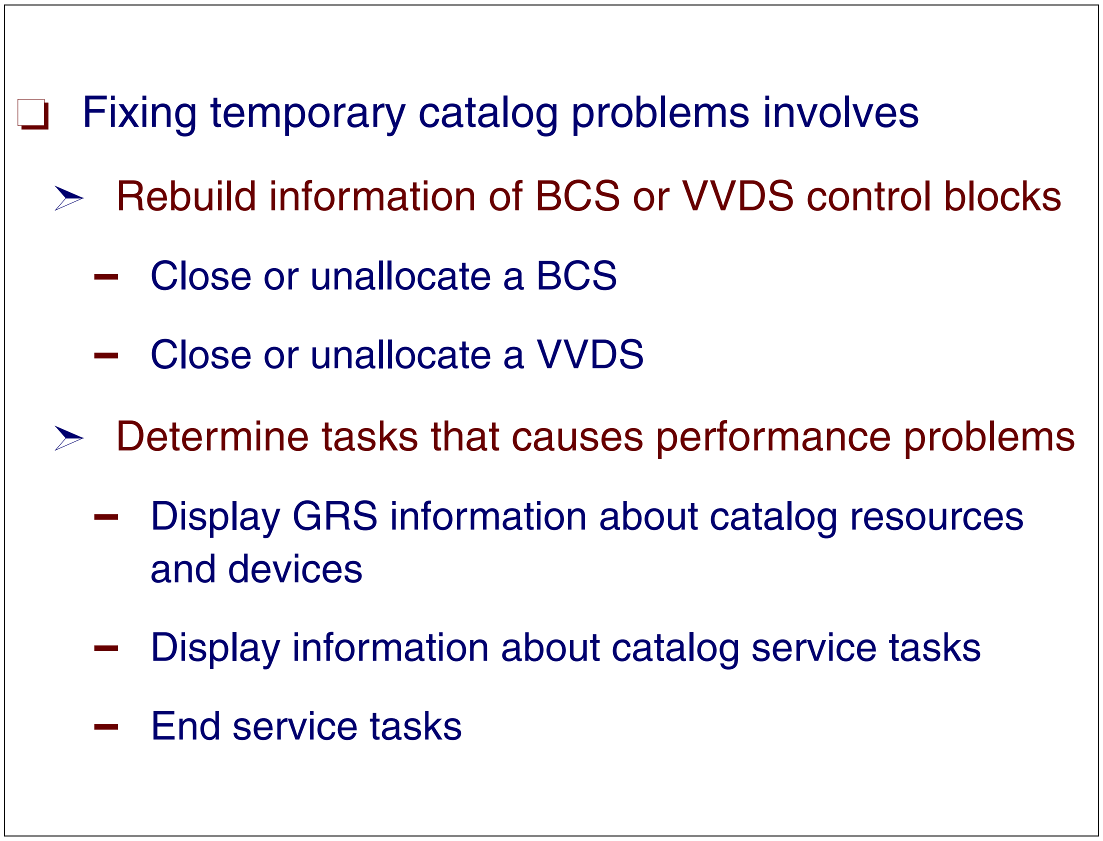
- **`MODIFY CATALOG,REPORT,CACHE`**

  The cache report lists cache type and statistics for the single catalogs which are open in the CAS.

- **`MODIFY CATALOG,RESTART`**

  Use this command to restart the catalog address space. This should be done *only* in error situations when the other option would be an IPL. See also 6.20, "Catalog address space (CAS)" on page 345.

6.22 Fixing temporary catalog problems



- Fixing temporary catalog problems involves
  - Rebuild information of BCS or VVDS control blocks
    - Close or unallocate a BCS
    - Close or unallocate a VVDS
  - Determine tasks that causes performance problems
    - Display GRS information about catalog resources and devices
    - Display information about catalog service tasks
    - End service tasks

Figure 6-35 Fixing temporary catalog problems

Fixing temporary catalog problems

In this section we discuss how to rebuild information in the catalog address space and how to get information about and recover from performance slowdowns.

Rebuild information about a BCS or VVDS

Occasionally, the control blocks for a catalog kept in the catalog address space might be damaged. You might think the catalog is damaged and in need of recovery, when only the control blocks need to be rebuilt. If the catalog appears damaged, try rebuilding the control blocks first. If the problem persists, recover the catalog.

You can use the following commands to close or unallocate a BCS or VVDS in the catalog address space. The next access to the BCS or VVDS reopens it and rebuilds the control blocks.

- `MODIFY CATALOG,CLOSE(catalogname)` - Closes the specified catalog but leaves it allocated.
- `MODIFY CATALOG,UNALLOCATE(catalogname)` - Unallocates a catalog; if you do not specify a catalog name, *all* catalogs are unallocated.
- `MODIFY CATALOG,VCLOSE(volser)` - Closes the VVDS for the specified volser.
- `MODIFY CATALOG,VUNALLOCATE` - Unallocates *all* VVDSs; you cannot specify a volser, so try to use VCLOSE first.

Recover from performance slow downs

Sometimes the performance of the catalog address space slows down. Catalog requests may take a long time or even hang. There can be various reasons for such situations, for example a volume on which a catalog resides is reserved by another system, thus all requests from your system to this catalog wait until the volume is released.

Generally, the catalog component uses two resources for serialization:

- SYSIGGV2 to serialize on the BCS
- SYSZVVDS to serialize on the VVDS

Delays or hangs can occur if the catalog needs one of these resources and it is held already by someone else, for example by a CAS of another system. You can use the following commands to display global resource serialization (GRS) data:

- **`D GRS,C`** - Display GRS contention data for all resources, who is holding a resource, and who is waiting.
- **`D GRS,RES=(resourcename)`** - Displays information for a specific resource.
- **`D GRS,DEV=devicenumber`** - Displays information about a specific device, such as whether it is reserved by the system.

You should route these commands to all systems in the sysplex to get an overview about hang situations.

When you have identified a catalog address space holding a resource for a long time, or the GRS outputs do not show you anything but you have still catalog problems, you can use the following command to get detailed information about the catalog services task:

- **`MODIFY CATALOG,LIST`** - Lists the currently active service tasks, their task IDs, duration, and the job name for which the task is handling the request.

You should watch for tasks with long duration time. You can get detailed information about a specific task by running the following command for a specific task ID:

- **`MODIFY CATALOG,LISTJ(taskid),DETAIL`** - Shows you detailed information about a service task, for example if it's waiting for the completion of an ENQ.

When you have identified a long running task which could be in a deadlock situation with another task (on another system), you can end and redrive the task to resolve the lockout. The following commands help you to end a catalog service task:

- **`MODIFY CATALOG,END(taskid),REDRIVE`** - End a service task and redrive it.
- **`MODIFY CATALOG,END(taskid),NOREDRIVE`** - Permanently end the task without redriving.
- **`MODIFY CATALOG,ABEND(taskid)`** - Abnormally end a task which could not be stopped by using the **`END`** parameter.

You can use the **`FORCE`** parameter for these commands if the address space that the service task is operating on behalf of has ended abnormally. Use this parameter *only* in this case.

You could also try to end the job for which the catalog task is processing a request.

For more information about the MODIFY CATALOG command and fixing temporary catalog problems, refer to *z/OS DFSMS: Managing Catalogs,* SC26-7409.

6.23 Enhanced catalog sharing

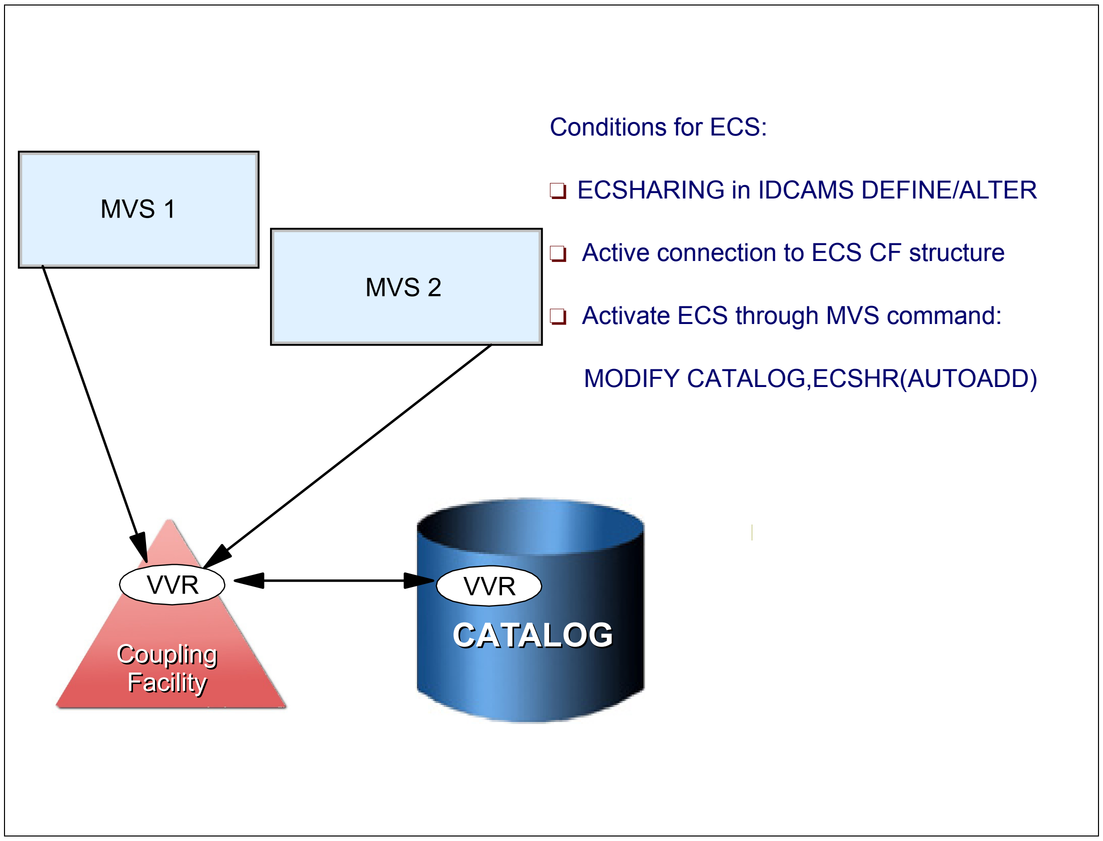

MVS 1
MVS 2
VVR
Coupling Facility
VVR
CATALOG
Conditions for ECS:
❑ ECSHARING in IDCAMS DEFINE/ALTER
❑ Active connection to ECS CF structure
❑ Activate ECS through MVS command:
MODIFY CATALOG,ECSHR(AUTOADD)

Figure 6-36 DFSMS enhanced catalog sharing

Enhanced catalog sharing (ECS)

In 6.9, "Sharing catalogs across systems" on page 320 we talked already about sharing catalogs across multiple systems. Catalog uses its specific VVR to serialize the access from multiple systems. To read serialization information from the VVR on the DASD causes a lot of overhead for I/O operations, though it is better to have this VVR than to discard the catalog buffers, when in shared catalog mode. This is called *VVDS mode*.

Most of the overhead associated with shared catalog is eliminated if you use *enhanced catalog sharing*. ECS uses a cache coupling facility structure to keep the special VVR. Also the coupling facility structure (as defined in CFRM) keeps a copy of updated records.

There is no I/O necessary to read the catalog VVR in order to verify the updates. In addition, the eventual modifications are also kept in the coupling facility structure, thus avoiding more I/O.

ECS saves about 50 percent in elapsed time and provides an enormous reduction in ENQ/Reserves.

Steps to implement enhanced catalog sharing

There are three steps you need to follow to implement ECS. In short, they are:

1. Define a coupling facility cache structure with name SYSIGGCAS_ECS in the CFRM couple data set and activate the this CFRM policy.

This action should connect all ECS-eligible systems to the ECS structure.

2. Define or alter your existing catalogs with the attribute ECSHARING using the IDCAMS **`DEFINE`** or **`ALTER`** commands.

   The ECSHARING attribute makes a catalog *eligible* for sharing using the ECS protocol, as opposed to the VVDS/VVR protocol. The catalog can still be used in VVDS mode.

3. Activate all eligible catalogs for ECS sharing.

   You can use the command **`MODIFY CATALOG,ECSHR(AUTOADD)`** on one system to activate all ECS-eligible catalogs throughout the sysplex for ECS sharing. They are automatically activated at their next reference. You can manually add a catalog to use ECS by running the command **`MODIFY CATALOG,ECSHR(ENABLE,catname)`** where *`catname`* is the name of the catalog you want to add.

Only those catalogs that were added are shared in ECS mode. The command **`MODIFY CATALOG,ECSHR(STATUS)`** shows you the ECS status for each catalog, if it is eligible or not, and if it is already activated.

Restrictions

The following restrictions apply to ECS mode usage:

- You cannot use ECS mode from one system and VVDS mode from another system simultaneously to share a catalog. You will get an error message if you try this.

  Attention: If you attempt to use a catalog that is currently ECS-active from a system *outside* the sysplex, the request might break the catalog.

- No more than 1024 catalogs can currently be shared using ECS from a single system.
- All systems sharing the catalog in ECS mode must have connectivity to the same Coupling Facility, and must be in the same global resource serialization (GRS) complex.
- When you use catalogs in ECS mode, convert the resource SYSIGGV2 to a SYSTEMS enqueue. Otherwise, the catalogs in ECS mode will be damaged.

For more information about ECS, refer to *z/OS DFSMS: Managing Catalogs,* SC26-7409.

For information about defining coupling facility structures, see *z/OS MVS Setting Up a Sysplex,* SA22-7625.

7

DFSMS Transactional VSAM Services

DFSMS Transactional VSAM Services (DFSMStvs) is an enhancement to VSAM RLS access that enables multiple batch update jobs and CICS to share access to the same data sets. DFSMStvs provides two-phase commit and backout protocols, as well as backout logging and forward recovery logging. DFSMStvs provides transactional recovery directly within VSAM.

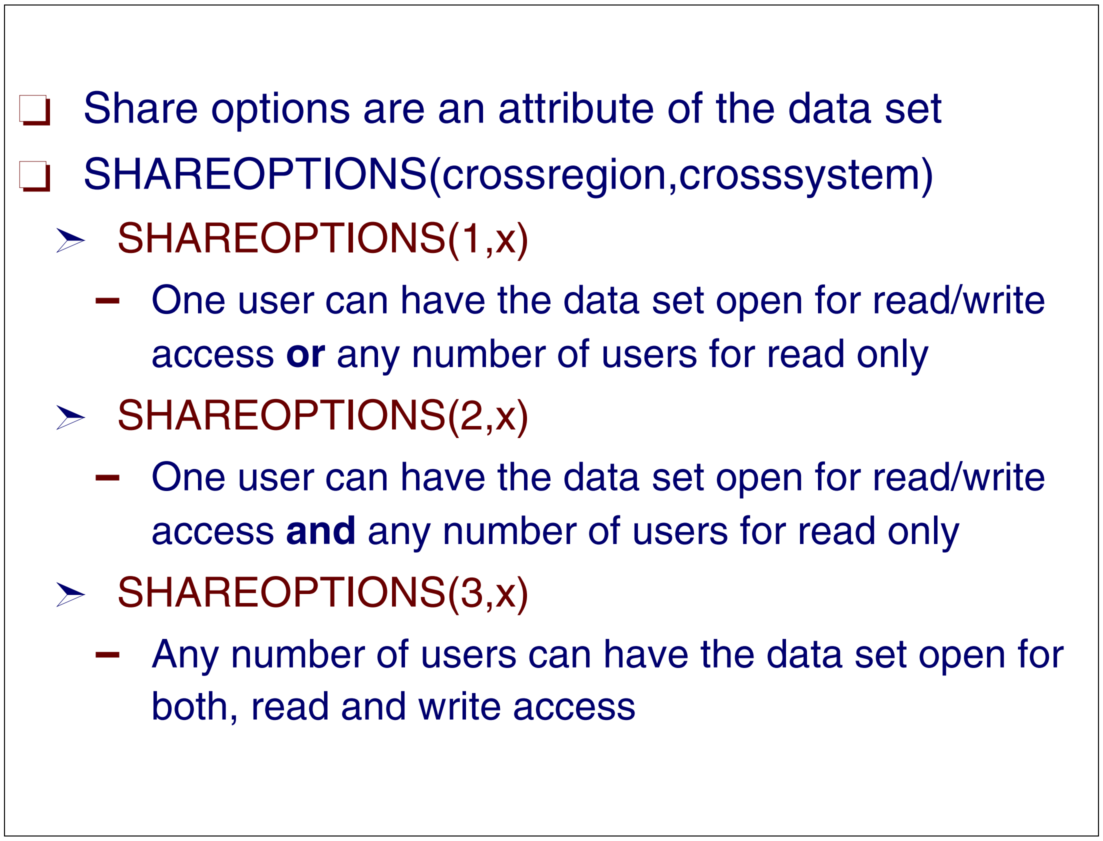
As an extension of VSAM RLS, DFSMStvm enables any job or application that is designed for data sharing to read-share or write-share VSAM recoverable data sets. VSAM RLS provides a server for sharing VSAM data sets in a sysplex. VSAM RLS uses coupling-facility-based locking and data caching to provide sysplex-scope locking and data access integrity, while DFSMStvs adds logging, commit, and backout processing.

To understand DFSMStvs, it is necessary to first review base VSAM information and VSAM record-level sharing (RLS).

In this chapter we cover the following topics:

- Review of base VSAM information and CICS concepts
- Introduction to VSAM RLS
- Introduction to DFSMStvs

7.1 VSAM share options



- Share options are an attribute of the data set
- SHAREOPTIONS(crossregion,crosssystem)
  - SHAREOPTIONS(1,x)
    - One user can have the data set open for read/write access **or** any number of users for read only
  - SHAREOPTIONS(2,x)
    - One user can have the data set open for read/write access **and** any number of users for read only
  - SHAREOPTIONS(3,x)
    - Any number of users can have the data set open for both, read and write access

Figure 7-1 VSAM share options

Share options as a data set attribute

The share options of a VSAM data set are specified as a parameter for an IDCAMS DEFINE CLUSTER that creates a new data set. They show how a component or cluster can be shared among users.

SHAREOPTIONS (crossregion,crosssystem)

The cross-region share options specify the amount of sharing allowed among regions within the same system or multiple systems. Cross-system share options specify how the data set is shared among systems. The serialization should be done by using global resource serialization (GRS) or a similar product.

SHAREOPTIONS (1,x)

The data set can be shared by any number of users for read access (open for input), or it can be accessed by only one user for read/write access (open for output). If the data set is open for output by one user, a read or read/write request by another user will fail. With this option, VSAM ensures complete data integrity for the data set. When the data set is already open for RLS processing, any request to open the data set for non-RLS access will fail.

SHAREOPTIONS (2,x)

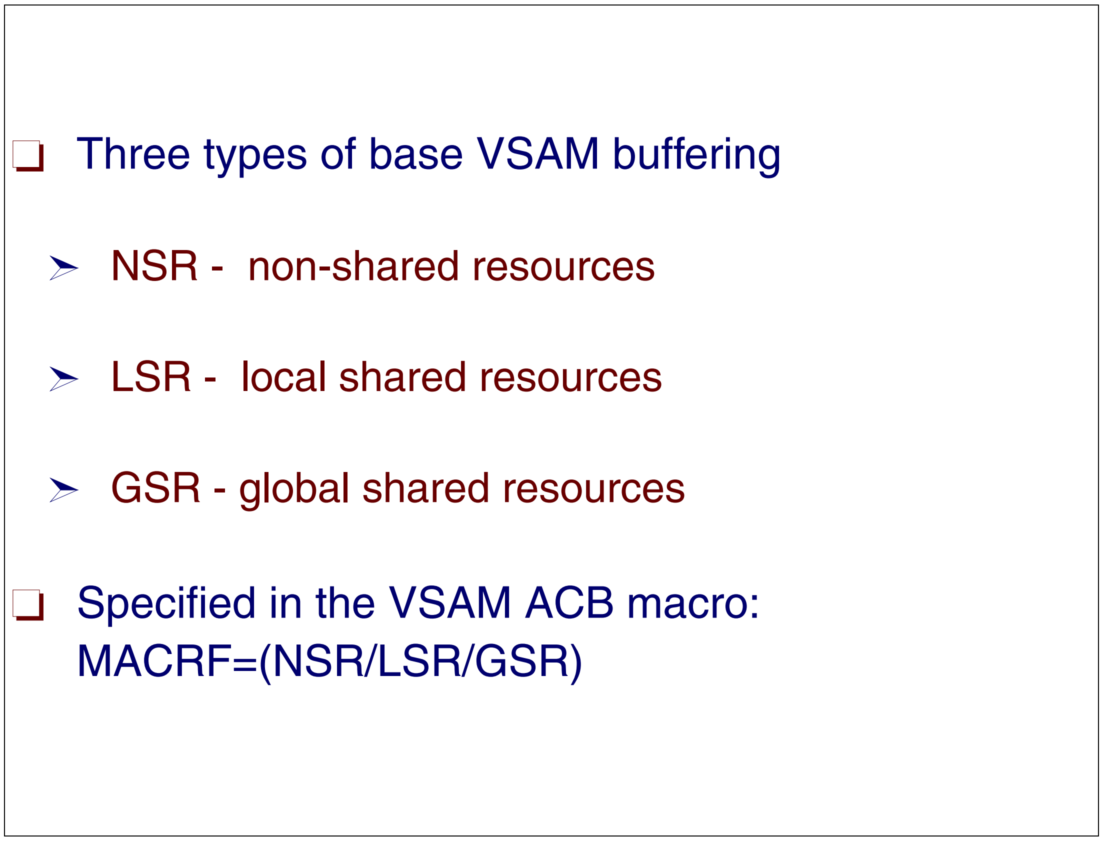
The data set can be shared by one user for read/write access, and by any number of users for read access. If the data set is open for output by one user, another open for output request will fail, whereas a request for read access will succeed. With this option, VSAM ensures write integrity. If the data set is open for RLS processing, non-RLS access for read is allowed. VSAM provides full read and write integrity for its RLS users, but no read integrity for non-RLS access.

SHAREOPTIONS (3,x)

The data set can be opened by any number of users for read and write request. VSAM does not ensure any data integrity. It is the responsibility of the users to maintain data integrity by using enqueue and dequeue macros. This setting does not allow any type of non-RLS access while the data set is open for RLS processing.

For more information about VSAM share options, refer to *z/OS DFSMS: Using Data Sets,* SC26-7410.

7.2 Base VSAM buffering



- Three types of base VSAM buffering
  - NSR - non-shared resources
  - LSR - local shared resources
  - GSR - global shared resources
- Specified in the VSAM ACB macro: MACRF=(NSR/LSR/GSR)

Figure 7-2 Base VSAM buffering

Three types of base VSAM buffering

Before VSAM RLS there were only three buffering techniques available for the user that opens the data set.

NSR - non-shared resources

For NSR, data buffers belong to a particular request parameter list (RPL). When an application uses an RPL (for example, for a direct GET request), VSAM manages the buffers as follows:

1. For record 1000, VSAM locates the CI containing the record and reads it into the local buffer in private storage.
2. If the next GET request is for record 5000, which is in a different CI from record 1000, VSAM overwrites the buffer with the new CI.
3. Another GET request for record 1001, which is in the same CI as record 1000, causes another I/O request for the CI to read it into the buffer, since it had been overlaid by the second request for record 5000.

LSR - local shared resources

For LSR, data buffers are *managed* so that the buffers will not be overlaid. Before opening the data set, the user builds a resource pool in private storage using the BLDVRP macro. The LSR processing is as follows:

1. A GET request for record 1000 reads in the CI containing the record in one buffer of the buffer pool.
2. The next GET request for record 5000, which is in a different CI, will read in the CI in another, different buffer in the same buffer pool.
3. A third GET request for record 1001 can be satisfied by using the CI that was read in for the request for record 1000.

GSR - global shared resources

GSR is the same concept as LSR. The only difference is, with GSR the buffer pool and VSAM control blocks are built in common storage and can be accessed by any address space in the system.

For more information about VSAM buffering techniques refer to 4.43, “VSAM: Buffering modes” on page 180.

MACRF=(NSR/LSR/GSR)

The Access Method Control block (ACB) describes an open VSAM data set. A subparameter for the ACB macro is MACRF, in which you can specify the buffering technique to be used by VSAM. For LSR and GSR, you need to run the BLDVRP macro before opening the data set to create the resource pool.

For information about VSAM macros, refer to *z/OS DFSMS: Macro Instructions for Data Sets,* SC26-7408.

7.3 Base VSAM locking

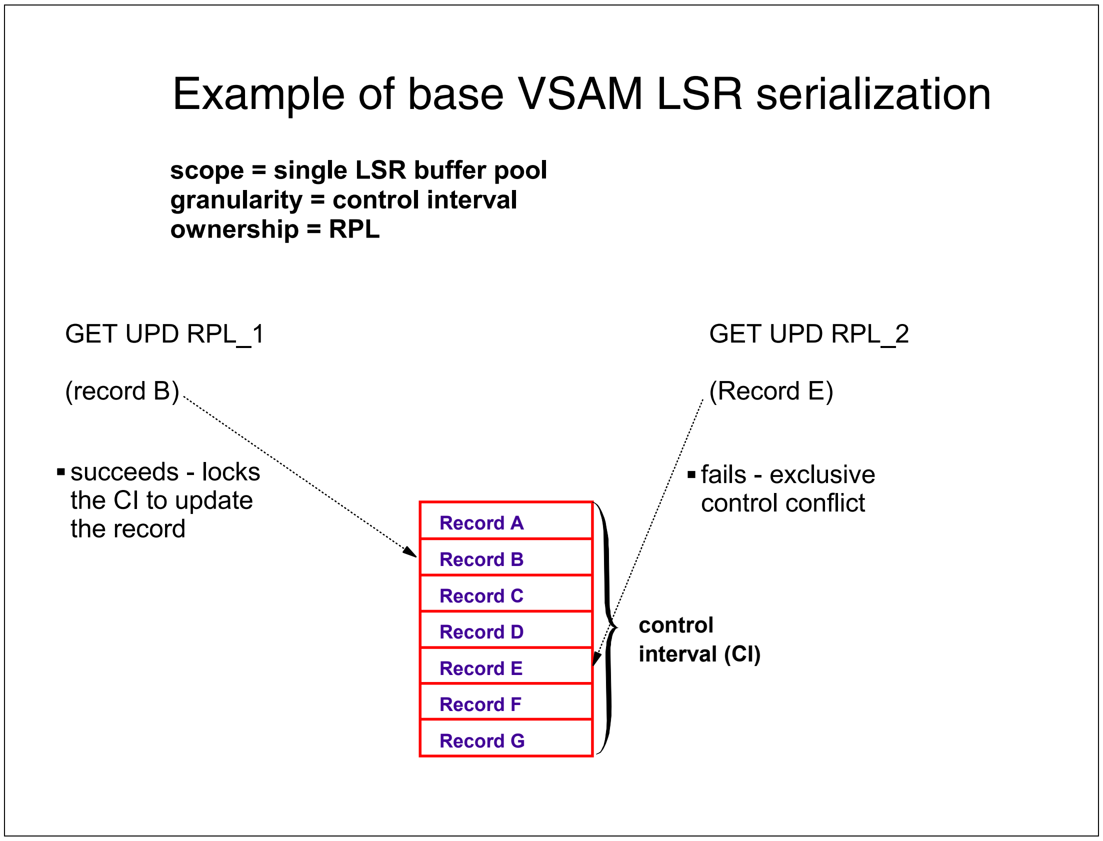

Example of base VSAM LSR serialization
scope = single LSR buffer pool
granularity = control interval
ownership = RPL
GET UPD RPL_1
(record B)
▪ succeeds - locks the CI to update the record
GET UPD RPL_2
(Record E)
▪ fails - exclusive control conflict
Record A
Record B
Record C
Record D
Record E
Record F
Record G
control interval (CI)

Figure 7-3 Example of LSR serialization

Serialization of base VSAM

Base VSAM serializes on a CI level. Multiple users attempting to access the same CI to read different records either defer on the CI or are returned an exclusive control conflict error by VSAM. Large CIs with many records per CI, or applications that repeatedly access the same CI, can have a performance impact due to retrying of exclusive control conflict errors or waiting on a deferred request.

Example of VSAM LSR serialization

In the example in Figure 7-3, two RPLs need to read and write different records in the same CI. RPL_1 gets access to the buffer containing the CI first and locks it for update processing against Record B. RPL_2 fails with an exclusive control conflict error on this CI although it needs to update a different record.

7.4 CICS function shipping before VSAM RLS

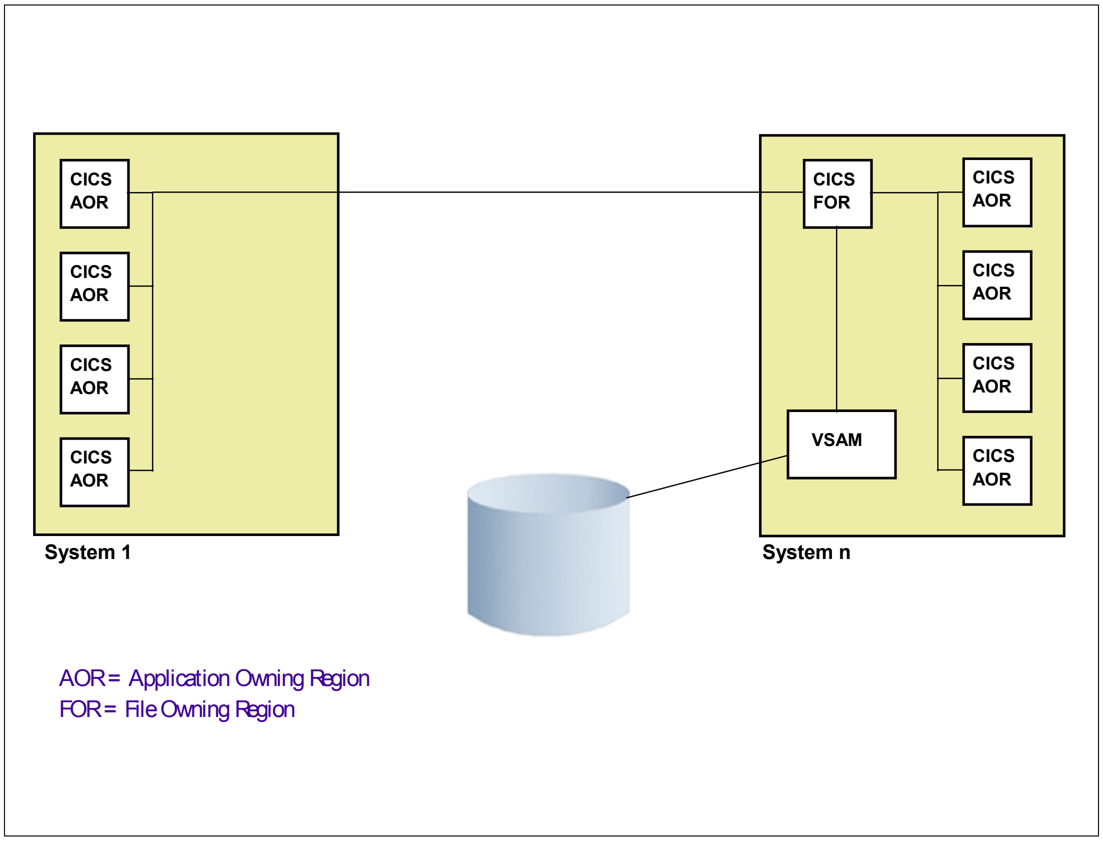


Figure 7-4 CICS function shipping before VSAM RLS

CICS function shipping

Prior to VSAM RLS, a *customer information control system* (CICS) VSAM data set was owned and directly accessed by one single CICS. Shared access across CICS Application Owning Regions (AORs) to a single VSAM data set was provided by CICS *function shipping*. With function shipping, one CICS File Owning Region (FOR) accesses the VSAM data sets on behalf of other CICS regions (see Figure 7-4).

Problems

There are a couple of problems with this kind of CICS configuration:

- CICS FOR is a single point of failure.
- Multiple system performance is not acceptable.
- Lack of scalability.

Over time the FORs became a bottleneck since the CICS environments became more and more complex. CICS required a solution to have direct shared access to VSAM data sets from multiple CICSs.

7.5 VSAM record-level sharing introduction

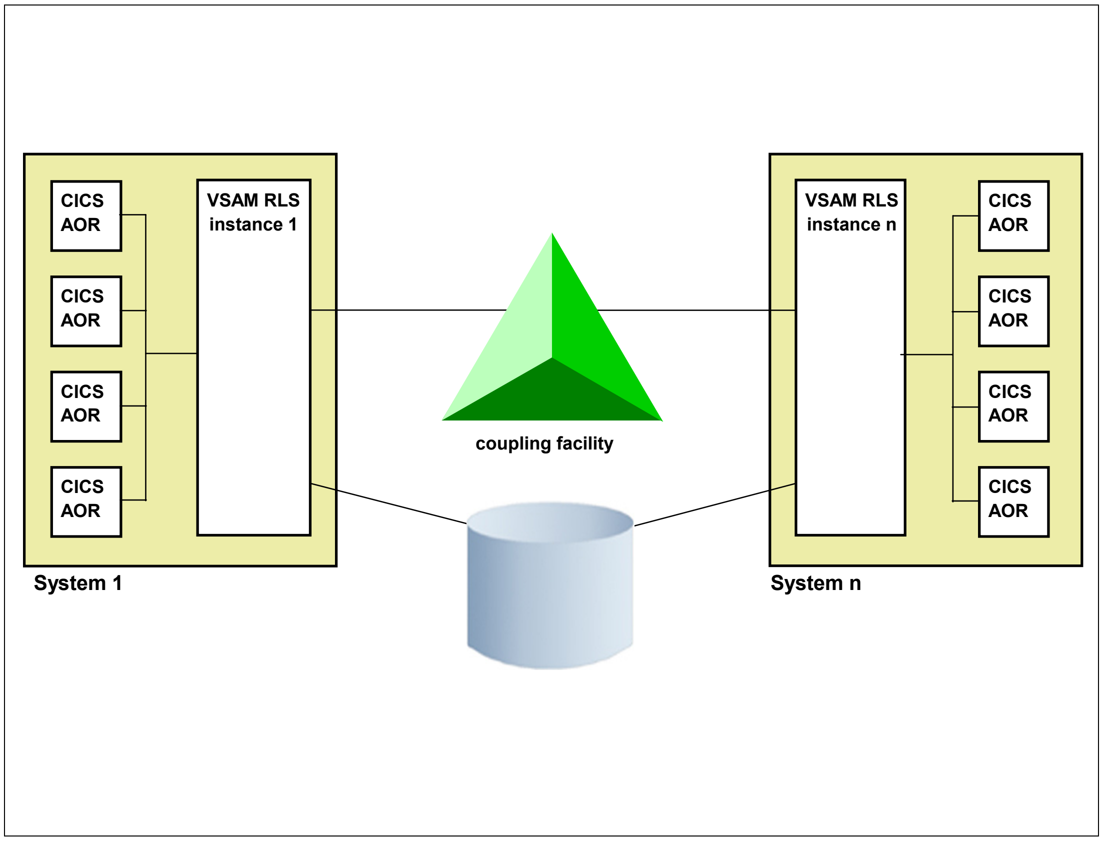


Figure 7-5 Parallel Sysplex CICS with VSAM RLS

Why VSAM record-level sharing (RLS)

The solution for the problems inherent in CICS function shipping is VSAM record-level sharing. This is a major extension to base VSAM, and while it was designed for use by CICS, it can be used by any application. It provides an environment where multiple CICSs can directly access a shared VSAM data set (Figure 7-5).

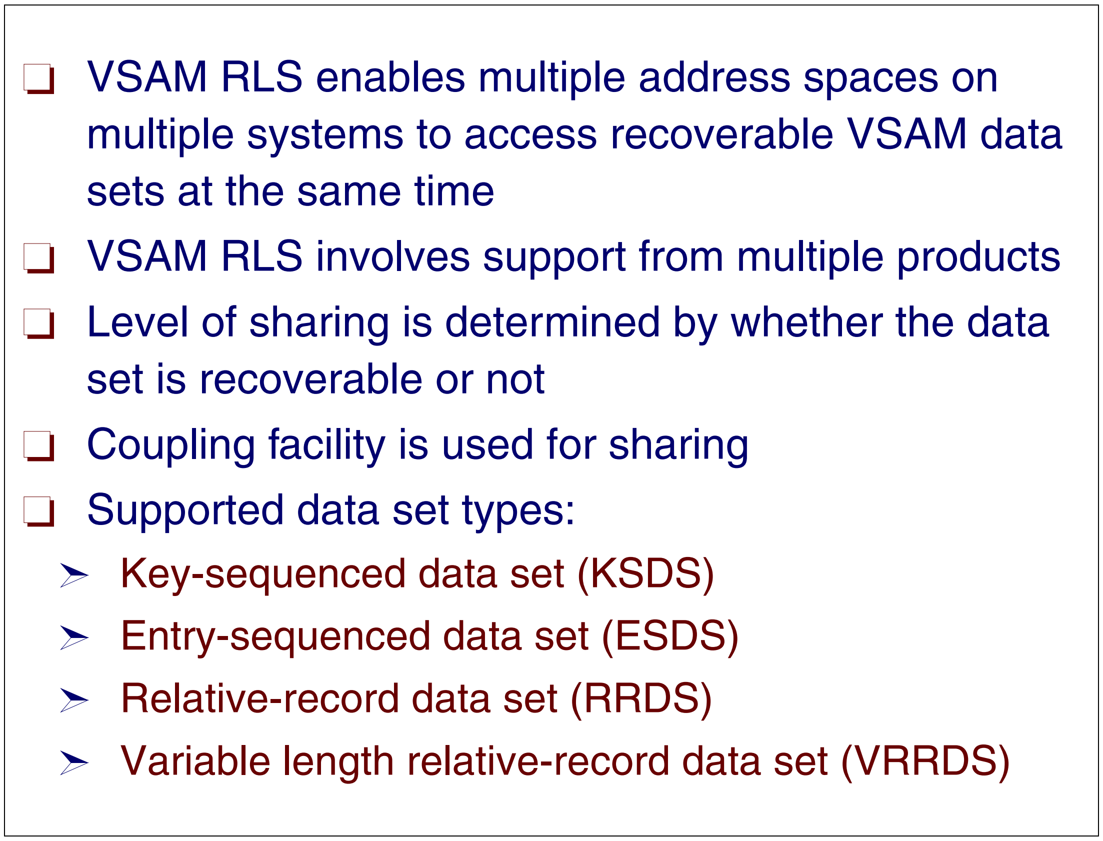
VSAM record-level sharing (RLS) is a method of access to your existing VSAM files that provides full read and write integrity at the *record level* to any number of users in your parallel sysplex.

Advantages of VSAM RLS

The advantages of VSAM RLS are:

- Enhances cross-system data sharing - scope is sysplex
- Improves performance and availability in CICS and also non-CICS VSAM environments
- Provides data protection after a system failure
- Provides automation for data recovery
- Provides full read/write integrity to your existing VSAM files; the user does not need to serialize using ENQ/DEQ macros
- Allows CICS to *register* as a *recoverable subsystem*, which will automate recovery processing while protecting the data records to be recovered

7.6 VSAM RLS overview



- ❑ VSAM RLS enables multiple address spaces on multiple systems to access recoverable VSAM data sets at the same time
- ❑ VSAM RLS involves support from multiple products
- ❑ Level of sharing is determined by whether the data set is recoverable or not
- ❑ Coupling facility is used for sharing
- ❑ Supported data set types:
  - ➢ Key-sequenced data set (KSDS)
  - ➢ Entry-sequenced data set (ESDS)
  - ➢ Relative-record data set (RRDS)
  - ➢ Variable length relative-record data set (VRRDS)

Figure 7-6 VSAM RLS overview

Multiple access on recoverable data sets

VSAM RLS is a data set access mode that enables multiple address spaces, CICS application-owning regions on multiple systems, and batch jobs to access recoverable VSAM data sets at the same time.

With VSAM RLS, multiple CICS systems can directly access a shared VSAM data set, eliminating the need to ship functions between the application-owning regions and file-owning regions. CICS provides the logging, commit, and backout functions for VSAM recoverable data sets. VSAM RLS provides record-level serialization and cross-system caching. CICSVR provides a forward recovery utility.

Multiple products involved

VSAM RLS processing involves support from multiple products:

- CICS Transaction Server
- CICS VSAM Recovery (CICSVR)
- DFSMS

Level of sharing

The level of sharing that is allowed between applications is determined by whether or not a data set is recoverable. For example:

- Both CICS and non-CICS jobs can have concurrent read or write access to nonrecoverable data sets. There is no coordination between CICS and non-CICS, so data integrity can be compromised.

- Non-CICS jobs can have read-only access to recoverable data sets concurrently with CICS jobs, which can have read or write access.

Coupling facility

The coupling facility (CF) is a shareable storage medium. It is licensed internal code (LIC) running in a special type of PR/SM™ logical partition (LPAR) in certain zSeries and S/390 processors. It can be shared by the systems in one sysplex only. A CF makes data sharing possible by allowing data to be accessed throughout a sysplex with assurance that the data will not be corrupted and that the data will be consistent among all sharing users.

VSAM RLS uses a coupling facility to perform data-set-level locking, record locking, and data caching. VSAM RLS uses the conditional write and cross-invalidate functions of the coupling facility cache structure, thereby avoiding the need for control interval (CI) level locking.

VSAM RLS uses the coupling facility caches as store-through caches. When a control interval of data is written, it is written to both the coupling facility cache and the direct access storage device (DASD). This ensures that problems occurring with a coupling facility cache do not result in the loss of VSAM data.

Supported data set types

VSAM RLS supports access to these types of data sets:

- Key-sequenced data set (KSDS)
- Entry-sequenced data set (ESDS)
- Relative-record data set (RRDS)
- Variable-length relative-record data set cluster (VRRDS)

VSAM RLS also supports access to a data set through an alternate index, but it does not support opening an alternate index directly in RLS mode. Also, VSAM RLS does not support access through an alternate index to data stored under z/OS UNIX System Services.

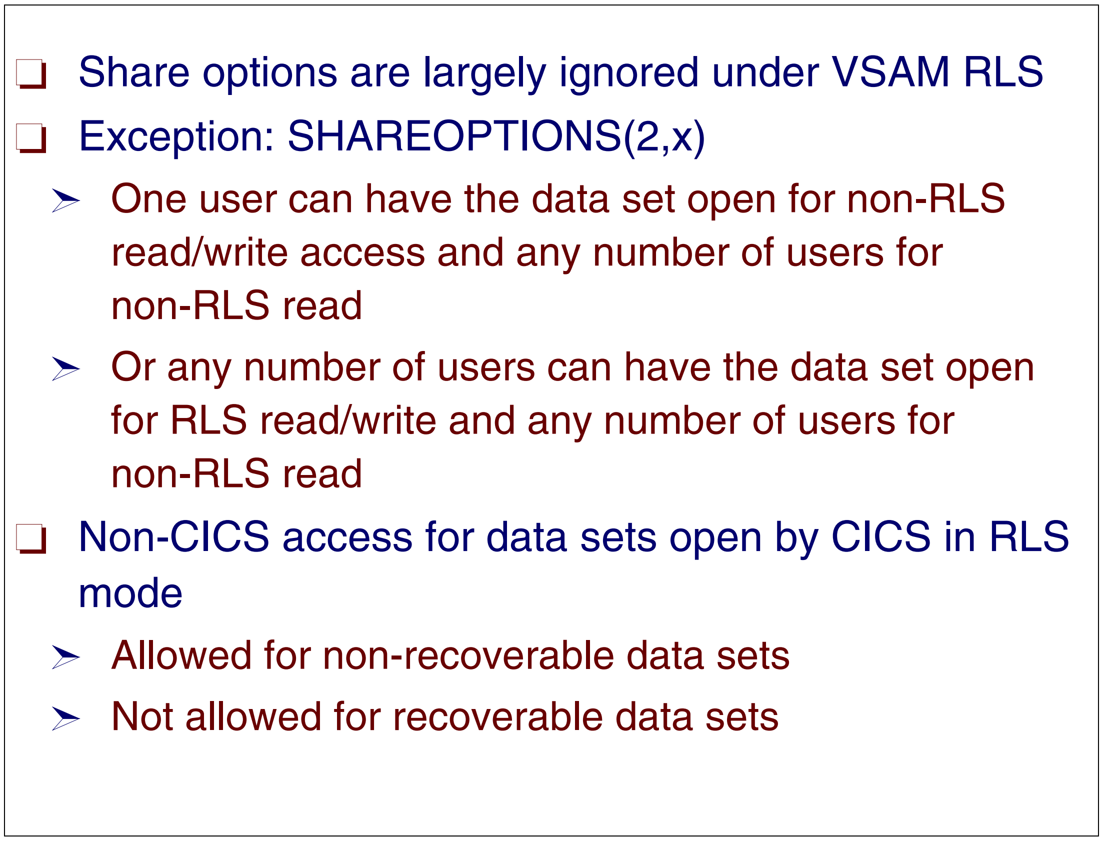
Extended format, extended addressability, and spanned data sets are supported with VSAM RLS. Compression is also supported.

VSAM RLS does not support:

- Linear data sets (LDS)
- Keyrange data sets
- KSDS with an imbedded index (defined with IMBED option)
- Temporary data sets
- Striped data sets
- Catalogs and VVDSs

Keyrange data sets and the IMBED attribute for a KSDS are obsolete. You cannot define new data sets as keyrange or with an imbedded index anymore. However, there still might be old data sets with these attributes in your installation.

7.7 Data set sharing under VSAM RLS



- ❑ Share options are largely ignored under VSAM RLS
- ❑ Exception: SHAREOPTIONS(2,x)
  - ➢ One user can have the data set open for non-RLS read/write access and any number of users for non-RLS read
  - ➢ Or any number of users can have the data set open for RLS read/write and any number of users for non-RLS read
- ❑ Non-CICS access for data sets open by CICS in RLS mode
  - ➢ Allowed for non-recoverable data sets
  - ➢ Not allowed for recoverable data sets

Figure 7-7 Data set sharing under VSAM RLS

Share options are largely ignored under VSAM RLS

The VSAM share options specification applies only when non-RLS access like NSR, LSR, or GSR is used. They are ignored when the data set is open for RLS access. Record-level sharing always assumes multiple readers and writers to the data set. VSAM RLS ensures full data integrity. When a data set is open for RLS access, non-RLS requests to open the data set fail.

Exception: SHAREOPTIONS(2,x)

For non-RLS access, SHAREOPTIONS(2,x) are handled as always. One user can have the data set open for read/write access and multiple users can have it open for read access only. VSAM does not provide data integrity for the readers.

If the data set is open for RLS access, non-RLS opens for *read* are possible. These are the only share options, where a non-RLS request to open the data set will not fail if the data set is already open for RLS processing. VSAM does not provide data integrity for the non-RLS readers.

Non-CICS access

RLS access from batch jobs to data sets that are open by CICS depends on whether the data set is recoverable or not. For recoverable data sets, non-CICS access from other applications (that do not act as recoverable resource manager) is not allowed.

See “VSAM RLS/CICS data set recovery” on page 368 for details.

7.8 Buffering under VSAM RLS

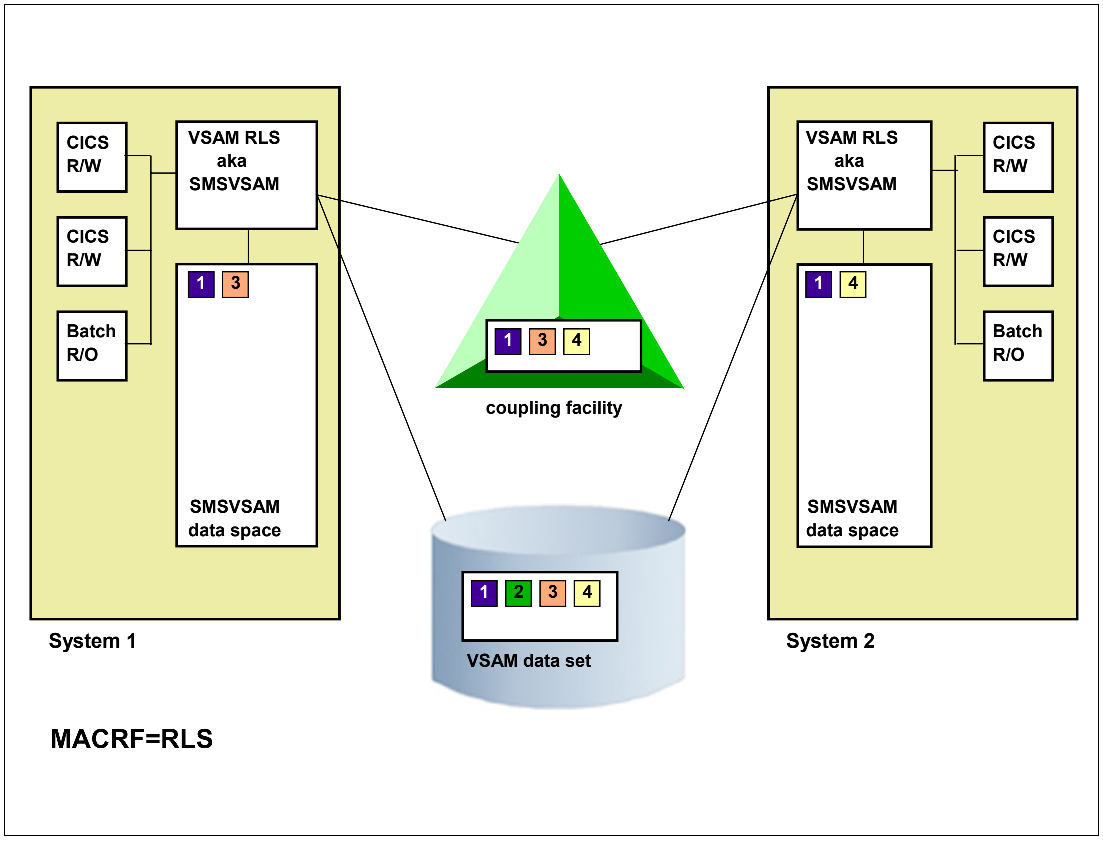


Figure 7-8 Buffering under VSAM RLS

New VSAM buffering technique MACRF=RLS

We have already discussed NSR, LSR, and GSR (refer to “Base VSAM buffering” on page 356). RLS is another method of buffering which you can specify in the MACRF parameter of the ACB macro. RLS and NSR/LSR/GSR are mutually exclusive.

Bufferpools in the data space

Unlike NSR, LSR, or GSR, the VSAM buffers reside in a data space and not in private or global storage of a user address space. Each image in the sysplex has one large local buffer pool in the data space.

The first request for a record after data set open for RLS processing will cause an I/O operation to read in the CI that contains this record. A copy of the CI is stored into the cache structure of the coupling facility and in the buffer pool in the data space.

Buffer coherency

Buffer coherency is maintained through the use of coupling facility (CF) cache structures and the XCF *cross-invalidation* function. For the example in Figure 7-8, that means:

1. System 1 opens the VSAM data set for read/write processing.
2. System 1 reads in CI1 and CI3 from DASD; both CIs are stored in the cache structure in the coupling facility.
3. System 2 opens the data set for read processing.

4. System 2 needs CI1 and CI4; CI1 is read from the CF cache, CI4 from DASD.
5. System 1 updates a record in CI1 and CI3; both copies of these CIs in the CF are updated.
6. XCF notices the change of these two CIs and invalidates the copy of CI1 for System 2.
7. System 2 needs another record from CI1; it notices that its buffer was invalidated and reads in a new copy of CI1 from the CF.

For further information about cross-invalidation refer to *z/OS MVS Programming: Sysplex Services Guide,* SA22-7617.

The VSAM RLS coupling facility structures are discussed in more detail in “Coupling facility structures for RLS sharing” on page 373.

7.9 VSAM RLS locking

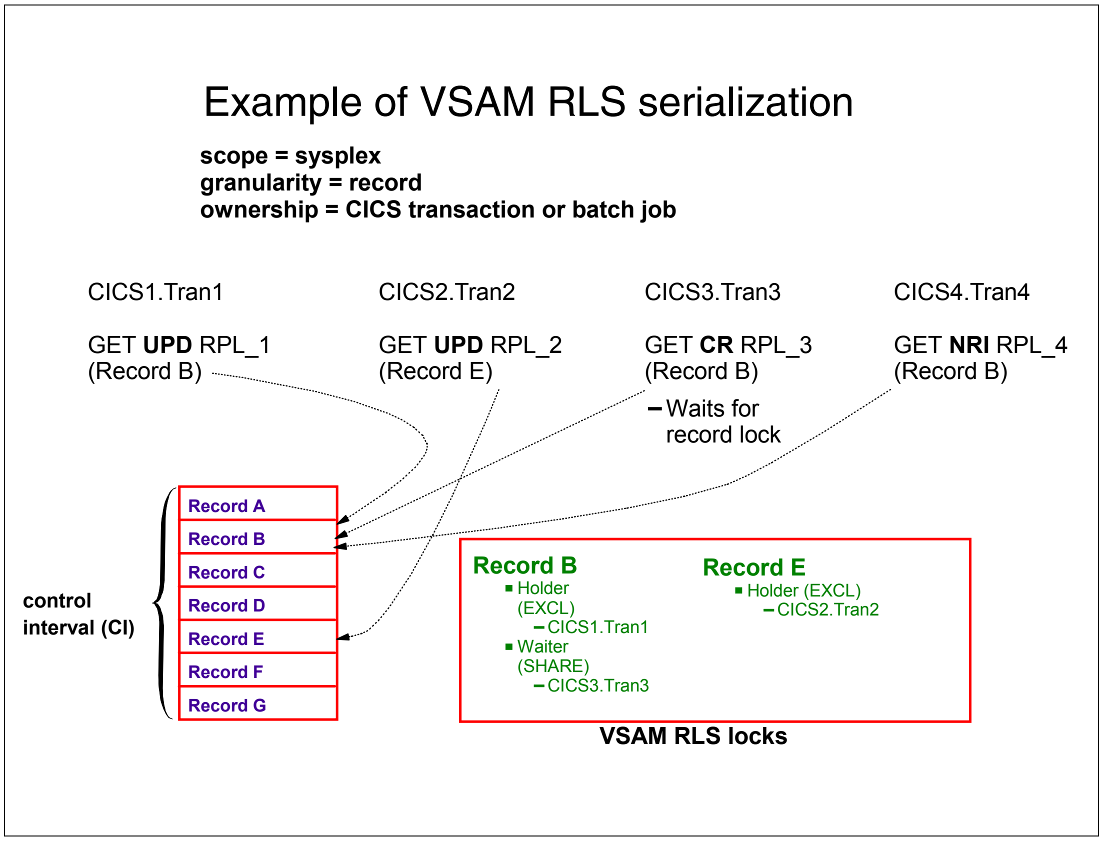


Figure 7-9 Example of VSAM RLS serialization

VSAM RLS serialization

Remember our example of LSR serialization in "Base VSAM locking" on page 358. The granularity under LSR, NSR, or GSR is a *control interval*, whereas VSAM RLS serializes on a *record level*. With VSAM RLS it is possible to concurrently update different records in the *same* control interval. Record locks for UPDATE are always exclusive. Record locks for read depend on the level of read integrity.

Three levels of read integrity

- NRI (no read integrity)

  This level tells VSAM not to obtain a record lock on the record accessed by a GET or POINT request. This avoids the overhead of record locking. This is sometimes referred to as a *dirty read* because the reader might see an uncommitted change made by another transaction.

  Even with this option specified, VSAM RLS still performs buffer validity checking and refreshes the buffer when the buffer is invalid.

- CR (consistent read)

  This level tells VSAM to obtain a *shared lock* on the record that is accessed by a GET or POINT request. It ensures that the reader does not see an uncommitted change made by another transaction. Instead, the GET or POINT request waits for the change to be committed or backed out. The request also waits for the exclusive lock on the record to be released.

- CRE (consistent read explicit)

  This level has a meaning similar to that of CR, except that VSAM RLS holds the shared lock on the record until the end of the unit of recovery, or unit of work. This option is only available to CICS or DFSMStvs transactions. VSAM RLS does not understand end-of-transaction for non-CICS or non-DFSMStvs usage.

The type of read integrity is specified either in the ACB macro or in the JCL DD statement:

- ACB RLSREAD=NRI/CR/CRE
- //dd1 DD dsn=datasetname,RLS=NRI/CR/CRE

Example

In our example in Figure 7-9 we have the following situation:

1. CICS transaction Tran1 gets an exclusive lock on Record B for update processing.
2. Transaction Tran2 gets an exclusive lock for update processing on Record E, which is in the same CI.
3. Transaction Tran3 needs a shared lock also on Record B for consistent read; it has to wait until the exclusive lock by Tran1 is released.
4. Transaction Tran4 does a dirty read (NRI); it doesn’t have to wait because in that case, no lock is necessary.

With NRI, Tran4 can read the record even though it is held exclusively by Tran1. There is no read integrity for Tran4.

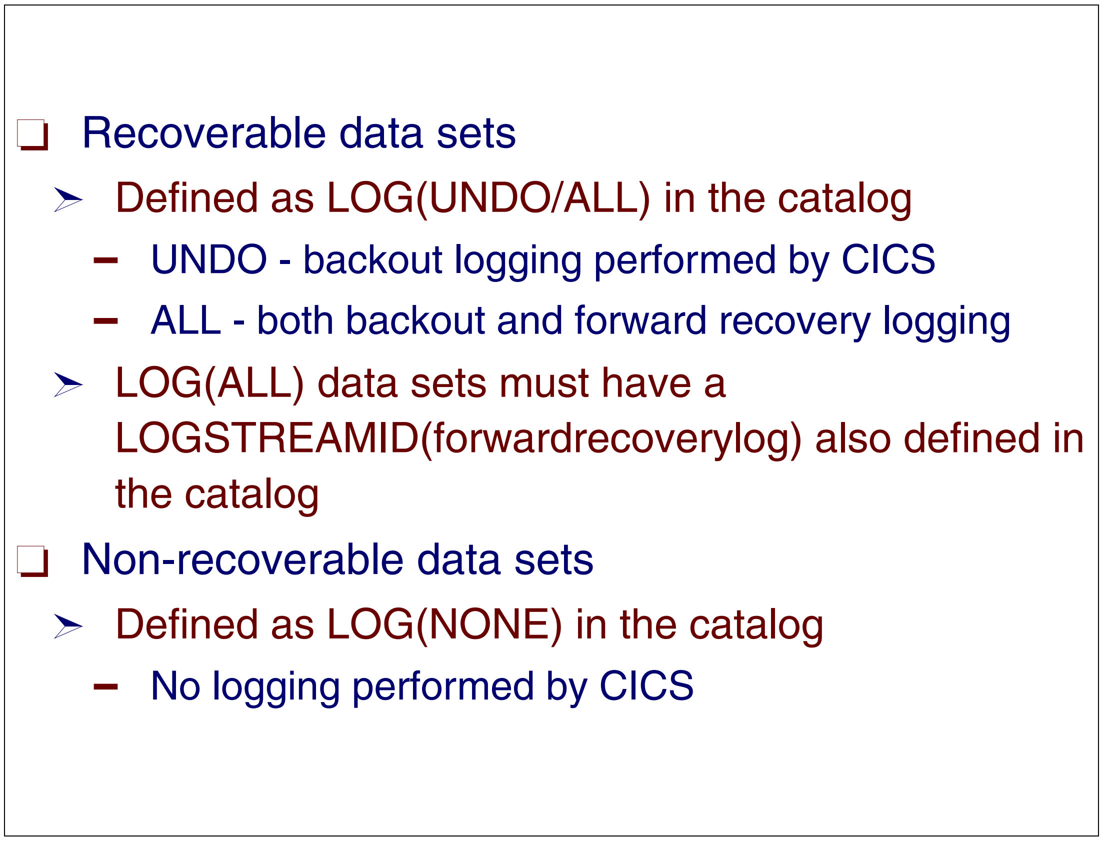
Coupling Facility

RLS locking is performed in the coupling facility through the use of a CF lock structure (IGWLOCK00) and the XES locking services.

Contention

When contention occurs on a VSAM record, the request that encountered the contention waits for the contention to be removed. The lock manager provides deadlock detection. When a lock request is in deadlock, the request is rejected, resulting in the VSAM record management request completing with a deadlock error response.

7.10 VSAM RLS/CICS data set recovery



- Recoverable data sets
  - Defined as LOG(UNDO/ALL) in the catalog
    - UNDO - backout logging performed by CICS
    - ALL - both backout and forward recovery logging
  - LOG(ALL) data sets must have a LOGSTREAMID(forwardrecoverylog) also defined in the catalog
- Non-recoverable data sets
  - Defined as LOG(NONE) in the catalog
    - No logging performed by CICS

Figure 7-10 Recoverable data sets

Recoverable data set

VSAM record-level sharing introduces a VSAM data set attribute called LOG. With this attribute a data set can be defined as *recoverable* or *non-recoverable*. A data set whose log parameter is undefined or NONE is considered non-recoverable. A data set whose log parameter is UNDO or ALL is considered recoverable. For recoverable data sets, a log of changed records is maintained to commit and back out transaction changes to a data set.

A data set is considered *recoverable* if the LOG attribute has one of the following values:

- UNDO

  The data set is *backward* recoverable. Changes made by a transaction that does not succeed (no commit was done) are backed out. CICS provides the *transactional recovery.* See also “Transactional recovery” on page 370.

- ALL

  The data set is both *backward* and *forward* recoverable. In addition to the logging and recovery functions provided for backout (transactional recovery), CICS records the image of changes to the data set, after they were made. The forward recovery log records are used by forward recovery programs and products such as CICS VSAM Recovery (CICSVR) to reconstruct the data set in the event of hardware or software damage to the data set. This is referred to as *data set recovery.* For LOG(ALL) data sets, both types of recovery are provided, transactional recovery and data set recovery.

For LOG(ALL) you need to define a logstream in which changes to the data sets are logged.

Non-recoverable data sets

A data set whose LOG parameter is undefined or NONE is considered as non-recoverable.

Non-CICS access to recoverable and non-recoverable data sets

VSAM RLS supports *non-recoverable* files. Non-recoverable means CICS does not do transactional recovery (logging, commit, backout). VSAM RLS provides record locking and file integrity across concurrently executing CICS and batch applications. Transactional recovery is *not* provided. This is because VSAM RLS does not provide undo logging and two-phase commit/backout support. Most transactions and batch jobs are *not* designed to use this form of data sharing.

Non-CICS read/write access for *recoverable* data sets that are open by CICS is not allowed. The recoverable attribute means that when the file is accessed in RLS mode, transactional recovery is provided. With RLS, the recovery is only provided when the access is through CICS file control, so RLS does *not* permit a batch (non-CICS) job to open a recoverable file for OUTPUT.

Transactional recovery is described in the next section.

7.11 Transactional recovery

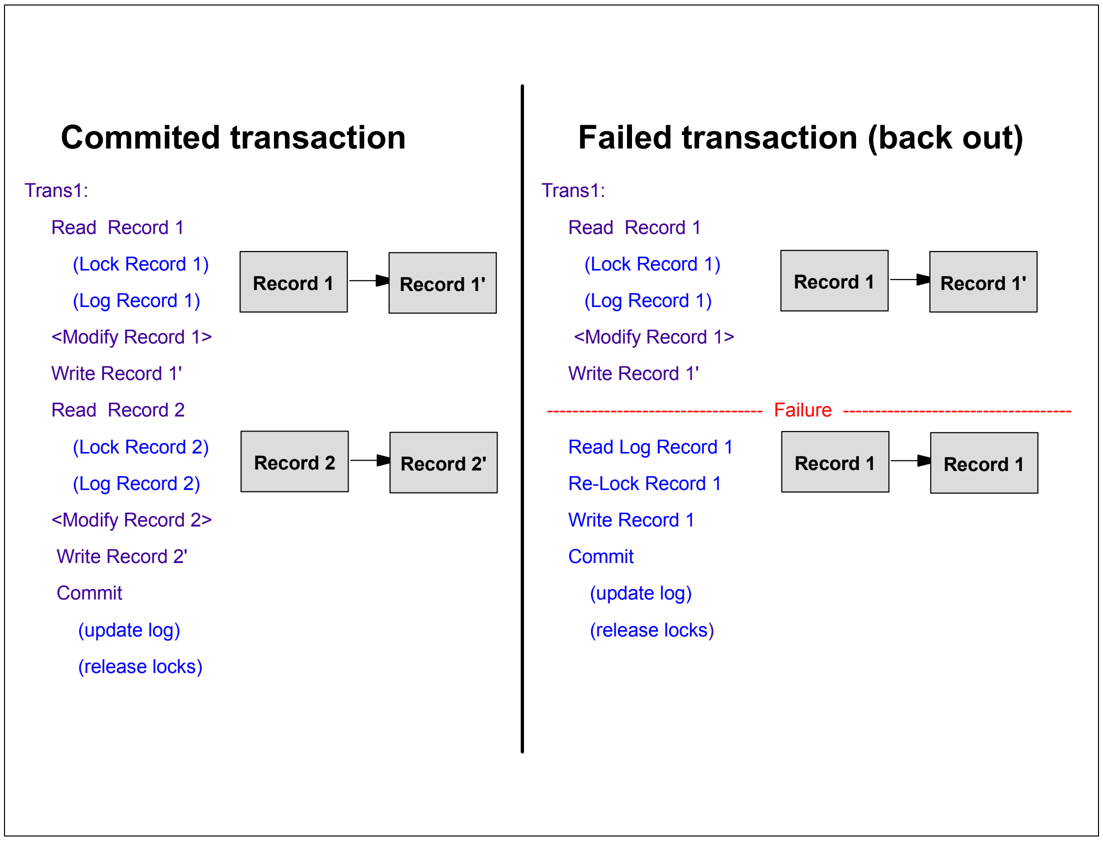


Figure 7-11 Transactional Recovery

CICS transactional recovery for VSAM recoverable data sets

During the life of a transaction, its changes to recoverable resources are *not* seen by other transactions. The exception is if you are using the no-read integrity (NRI) option. Then you might see uncommitted changes.

Exclusive locks that VSAM RLS holds on the modified records cause other transactions that have read-with-integrity requests and write requests for these records to wait. After the modifying transaction is committed or backed out, VSAM RLS releases the locks and the other transactions can access the records.

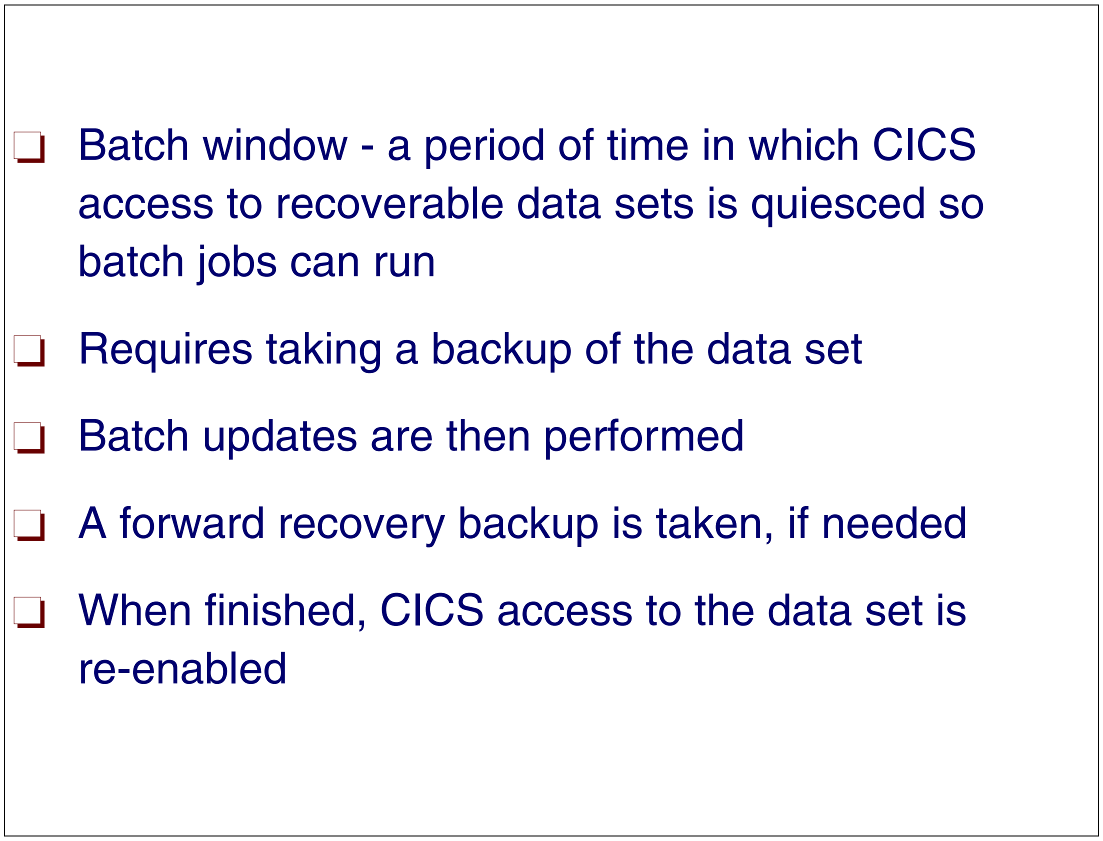
If the transaction fails, its changes are backed out. This capability is called *transactional recovery*.

The CICS backout function removes changes made to the recoverable data sets by a transaction. When a transaction abnormally ends, CICS performs a backout implicitly.

Example

In our example in Figure 7.11, transaction Trans1 is complete (committed) after Record 1 *and* Record 2 are updated. Transactional recovery ensures that either both changes are made or neither change is made. When the application requests *commit*, both changes are made atomically. In the case of an failure after updating Record 1, the change to this record is backed out. This applies only for recoverable data sets, not for non-recoverable ones.

7.12 The batch window problem



- Batch window - a period of time in which CICS access to recoverable data sets is quiesced so batch jobs can run
- Requires taking a backup of the data set
- Batch updates are then performed
- A forward recovery backup is taken, if needed
- When finished, CICS access to the data set is re-enabled

Figure 7-12 Batch window problem

Batch window

The batch window is a period of time in which online access to recoverable data sets must be disabled. During this time, no transaction processing can be done. This is normally done because it is necessary to run batch jobs or other utilities that do not properly support recoverable data, even if those utilities use also RLS access. Therefore, to allow these jobs or utilities to safely update the data, it is first necessary to make a copy of the data. In the event that the batch job or utility fails or encounters an error, this copy can be safely restored and online access can be re-enabled. If the batch job completes successfully, the updated copy of the data set can be safely used because only the batch job had access to the data while it was being updated. Therefore, the data cannot have been corrupted by interference from online transaction processing.

Quiescing a data set from RLS processing

Before you update a recoverable data set in *non-RLS* mode you should *quiesce* the data set around the sysplex. This is to ensure that no RLS access can be done while non-RLS applications are updating those data sets. The quiesced state is stored in the ICF catalog. After a quiesce has completed, all CICS files associated with the data set are *closed*. A quiesced data set can be opened in non-RLS mode only if no retained locks are presented. Once the data set was quiesced from RLS processing, it can be opened again in RLS mode only after it is *unquiesced*.

See "Interacting with VSAM RLS" on page 388 for information about how to quiesce and unquiesce a data set.

7.13 VSAM RLS implementation

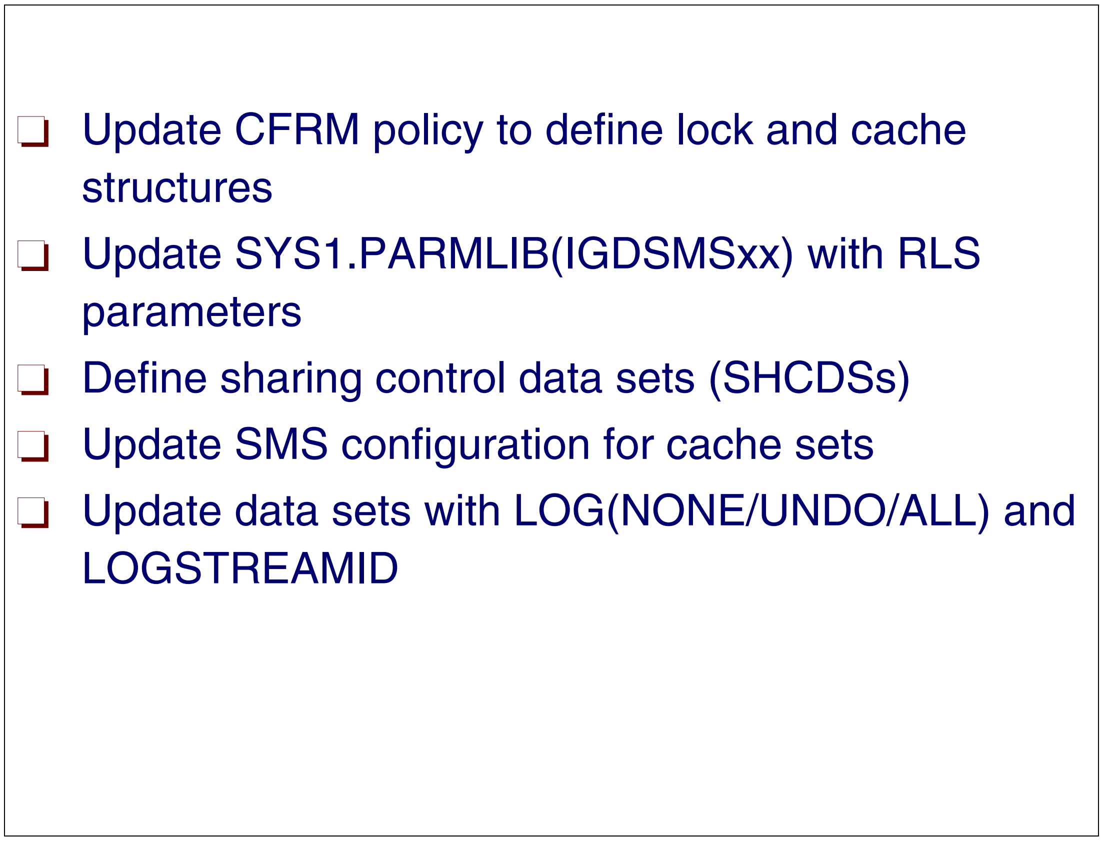

- ❑ Update CFRM policy to define lock and cache structures
- ❑ Update SYS1.PARMLIB(IGDSMSxx) with RLS parameters
- ❑ Define sharing control data sets (SHCDSs)
- ❑ Update SMS configuration for cache sets
- ❑ Update data sets with LOG(NONE/UNDO/ALL) and LOGSTREAMID

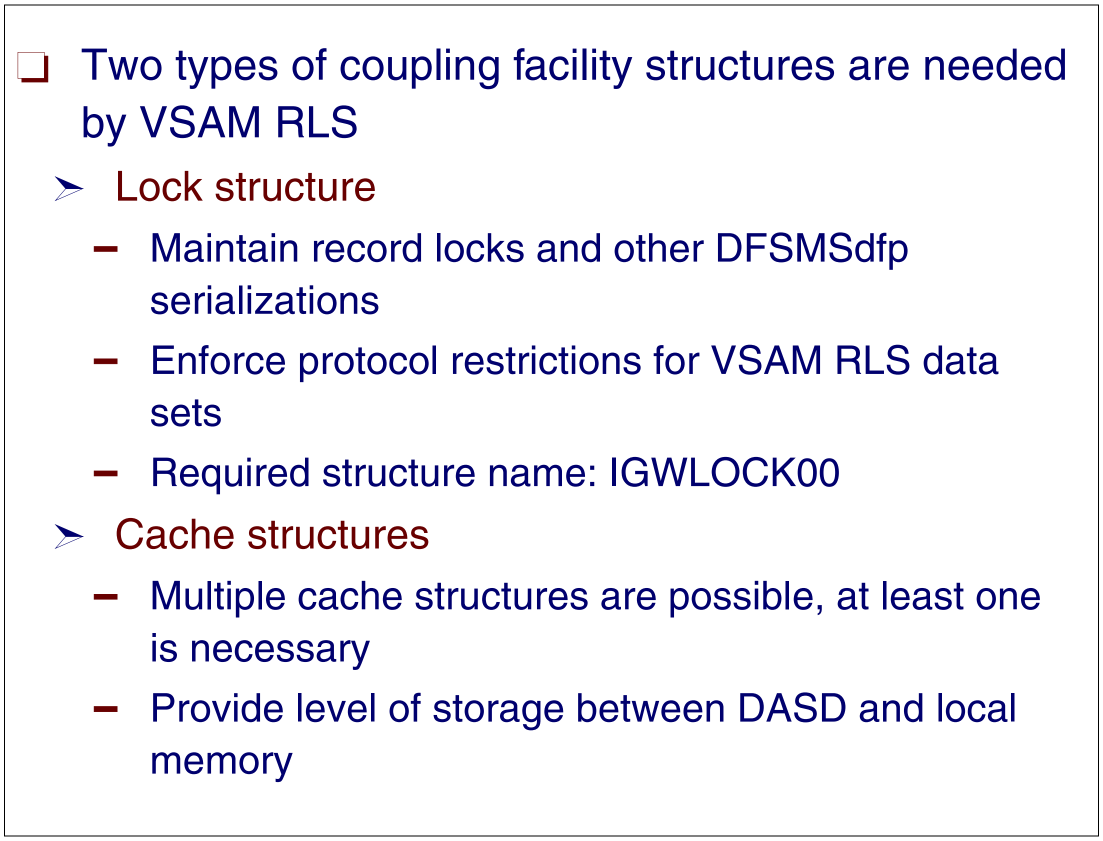
Figure 7-13 VSAM RLS configuration changes

VSAM RLS configuration changes

There are a couple of configuration changes necessary to run VSAM RLS. They include:

- Update coupling facility resource manager (CFRM) policy to define lock and cache structures.
- Update SYS1.PARMLIB(IGDSMSxx) with VSAM RLS parameters.
- Define new sharing control data sets (SHCDSs).
- Update SMS configuration for cache sets and assign them to a storage group.
- Update data sets with the attribute LOG(NONE/UNDO/ALL) and optionally assign a LOGSTREAMID.

7.14 Coupling facility structures for RLS sharing



- Two types of coupling facility structures are needed by VSAM RLS
  - Lock structure
    - Maintain record locks and other DFSMSdfp serializations
    - Enforce protocol restrictions for VSAM RLS data sets
    - Required structure name: IGWLOCK00
  - Cache structures
    - Multiple cache structures are possible, at least one is necessary
    - Provide level of storage between DASD and local memory

Figure 7-14 VSAM RLS coupling facility structures

Structures in a coupling facility

A CF stores information in structures. z/OS recognizes three structure types: cache, list, and lock. It provides a specific set of services for each of the structure types to allow the manipulation of data within the structure. VSAM RLS needs list and lock structures for data sharing and high-speed serialization.

Lock structure

In a parallel sysplex you need only *one* lock structure for VSAM RLS because only one VSAM sharing group is permitted. The required name is *IGWLOCK00*.

The lock structure is used to:

- Enforce the protocol restrictions for VSAM RLS data sets
- Maintain the record-level locks and other DFSMSdfp serializations

Ensure that the coupling facility lock structure has universal connectivity so that it is accessible from all systems in the parallel sysplex that support VSAM RLS.

Tip: For high-availability environments, use a nonvolatile coupling facility for the lock structure. If you maintain the lock structure in a volatile coupling facility, a power outage could cause a failure and loss of information in the coupling facility lock structure.

Cache structures

Coupling facility cache structures provide a level of storage hierarchy between local memory and DASD cache.

They are also used as system buffer pool with *cross-invalidation* being done (see "Buffering under VSAM RLS" on page 364).

Each coupling facility cache structure is contained in a single coupling facility. You may have multiple coupling facilities and multiple cache structures.

The minimum size of the cache structure should be 10 MB.

Sizing the lock and cache structure

For information about sizing the CF lock and cache structure for VSAM RLS refer to:

- *z/OS DFSMStvs Planning and Operation Guide,* SC26-7348
- *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402

There is also a tool available called CFSIZER. It is available on the IBM Web site at:

`http://www-1.ibm.com/servers/eserver/zseries/cfsizer/vsamrls.html`

Defining coupling facility structures

Use CFRM policy definitions to specify an initial and maximum size for each coupling facility structure. DFSMS uses the initial structure size you specify in the policy each time it connects to a coupling facility cache structure.

```
//STEP10   EXEC PGM=IXCMIAPU
//SYSPRINT DD   SYSOUT=A
//SYSABEND DD   SYSOUT=A
//SYSIN    DD   *
      DATA TYPE(CFRM) REPORT(YES)
      DEFINE POLICY NAME(CFRM01) REPLACE(YES)
        STRUCTURE NAME(CACHE01)
                  SIZE(70000)
                  INITSIZE(50000)
                  PREFLIST(CF01,CF02)
        STRUCTURE NAME(CACHE02)
                  SIZE(70000)
                  INITSIZE(50000)
                  PREFLIST(CF01,CF02)
        STRUCTURE NAME(IGWLOCK00)
                  SIZE(30000)
                  INITSIZE(15000)
                  PREFLIST(CF01,CF02)
/*
```

Figure 7-15 Example of defining VSAM RLS CF structure

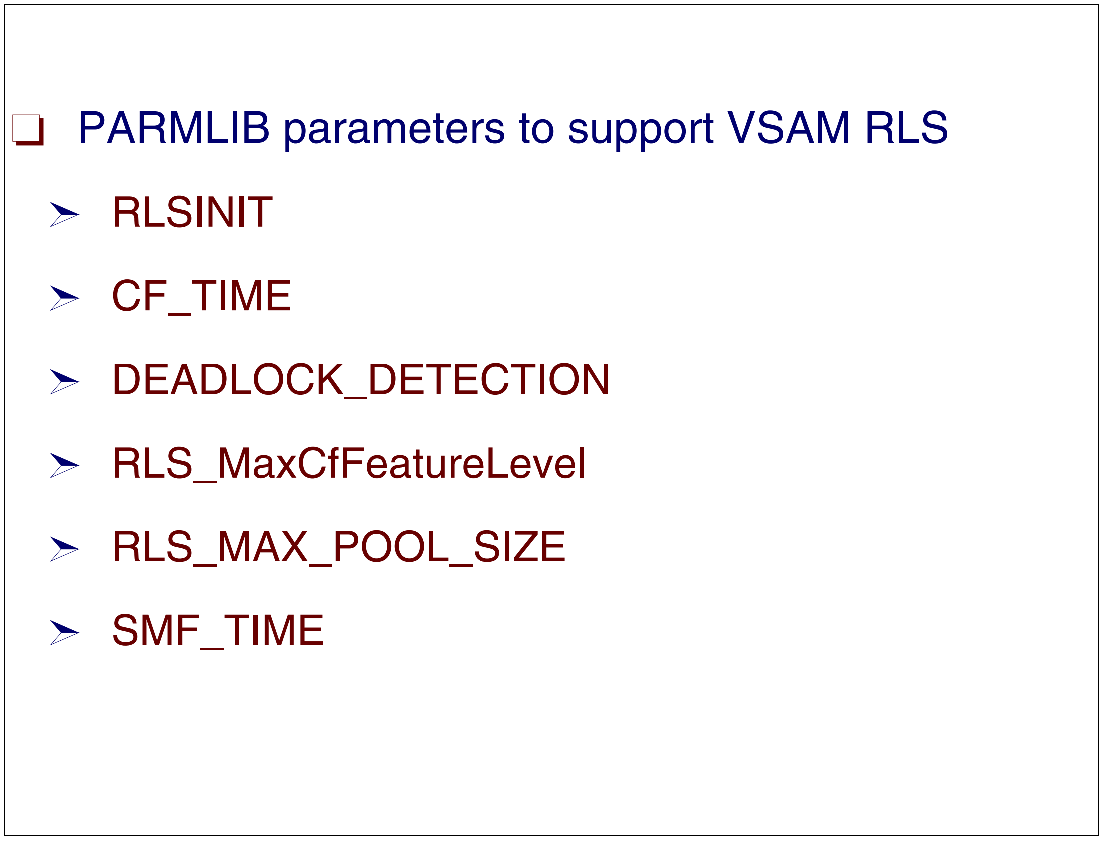
Displaying coupling facility structures

You can use the system command **DISPLAY XCF** to view your currently defined coupling facility structures. An example of XCF display of the lock structure is in Figure 7-16.

```
D XCF,STR,STRNAME=IGWLOCK00
IXC360I  10.00.38  DISPLAY XCF 337
STRNAME: IGWLOCK00
 STATUS: ALLOCATED
 TYPE: LOCK
 POLICY INFORMATION:
  POLICY SIZE    : 28600 K
  POLICY INITSIZE: 14300 K
  POLICY MINSIZE : 0 K
  FULLTHRESHOLD  : 80
  ALLOWAUTOALT   : NO
  REBUILD PERCENT: 75
  DUPLEX         : ALLOWED
  PREFERENCE LIST: CF1      CF2
  ENFORCEORDER   : NO
  EXCLUSION LIST IS EMPTY

 ACTIVE STRUCTURE
 ----------------
  ALLOCATION TIME: 02/24/2005 14:22:56
  CFNAME         : CF1
  COUPLING FACILITY: 002084.IBM.02.000000026A3A
                     PARTITION: 1F   CPCID: 00
  ACTUAL SIZE    : 14336 K
  STORAGE INCREMENT SIZE: 256 K
  ENTRIES:  IN-USE:          0 TOTAL:      33331,   0% FULL
  LOCKS:     TOTAL:    2097152
  PHYSICAL VERSION: BC9F02FD EDC963AC
  LOGICAL  VERSION: BC9F02FD EDC963AC
  SYSTEM-MANAGED PROCESS LEVEL: 8
  XCF GRPNAME    : IXCLO001
  DISPOSITION    : KEEP
  ACCESS TIME    : 0
  NUMBER OF RECORD DATA LISTS PER CONNECTION: 16
  MAX CONNECTIONS: 4
  # CONNECTIONS  : 4

 CONNECTION NAME  ID VERSION  SYSNAME  JOBNAME  ASID STATE
 ---------------- -- -------- -------- -------- ---- ----------------
 SC63             01 000100B0 SC63     SMSVSAM  0009 ACTIVE
 SC64             02 000200C6 SC64     SMSVSAM  000A ACTIVE
 SC65             03 000300DD SC65     SMSVSAM  000A ACTIVE
 SC70             04 00040035 SC70     SMSVSAM  000A ACTIVE
```

Figure 7-16   Example of XCF display of Structure IGWLOCK00

7.15 Update PARMLIB with VSAM RLS parameters



- PARMLIB parameters to support VSAM RLS
  - RLSINIT
  - CF_TIME
  - DEADLOCK_DETECTION
  - RLS_MaxCfFeatureLevel
  - RLS_MAX_POOL_SIZE
  - SMF_TIME

Figure 7-17 PARMLIB parameters to support VSAM RLS

New PARMLIB parameters to support VSAM RLS

The SYS1.PARMLIB member IGDSMSxx includes several parameters that support the coupling facility. With the exception of RLSINIT, these parameters apply across all systems in the parallel sysplex. The parameter values specified for the first system that was activated in the sysplex are used by all other systems in the sysplex.

The following IGDSMSxx parameters support VSAM RLS:

- RLSINIT({NO|YES})

  Specifies whether to start the SMSVSAM address space as part of system initialization
- CF_TIME(nnn|3600)

  Indicates the number of seconds between recording SMF type 42 records with subtypes 15, 16, 17, 18, and 19 for the CF (both cache and lock structures)
- DEADLOCK_DETECTION(iiii|15,kkkk|4)
  - Specifies the deadlock detection intervals used by the Storage Management Locking Services
  - iiii - Local detection interval in seconds
  - kkkk - Global detection interval, number of iterations of the local detection interval that must be run until the global deadlock detection is invoked

- RLS_MaxCfFeatureLevel({A|Z})

  Specifies the method that VSAM RLS uses to determine the size of the data that is placed in the CF cache structure
- RLS_MAX_POOL_SIZE({nnnn|100})

  Specifies the maximum size in megabytes of the SMSVSAM local buffer pool
- SMF_TIME({YES|NO})

  Specifies that the SMF type 42 records are created at the SMF interval time, and that all of the indicated records are synchronized with SMF and RMF data intervals

For more information about VSAM RLS parameters refer to *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

7.16 Define sharing control data sets

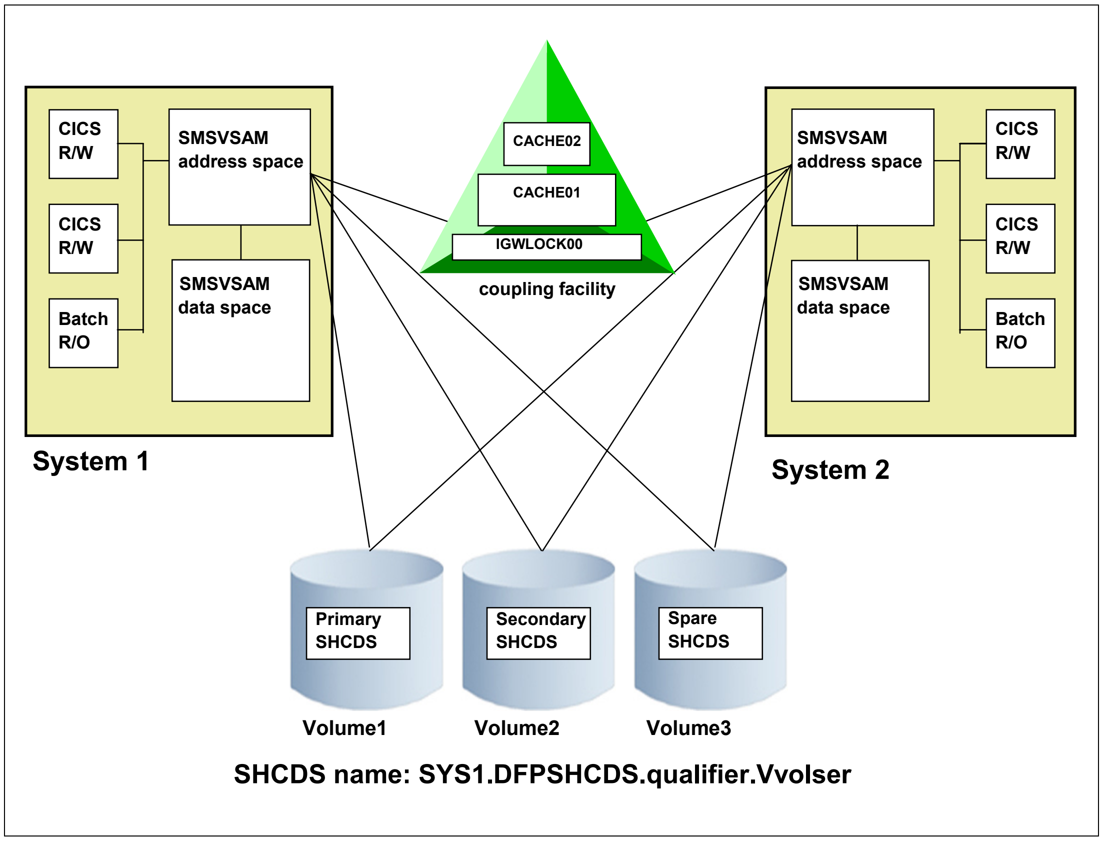


Figure 7-18 VSAM RLS sharing control data sets

VSAM RLS sharing control data sets

The sharing control data set (SHCDS) is designed to contain the information required for DFSMS to continue processing with a minimum of unavailable data and no corruption of data when failures occur. The SCDS data can be either SMS or non-SMS managed.

The SHCDS contains the following:

- Name of the CF lock structure in use
- System status for each system or failed system instance
- Time that the system failed
- List of subsystems and their status
- List of open data sets using the CF
- List of data sets with unbound locks
- List of data sets in permit non-RLS state

Defining sharing control data sets

The SHCDS is a logically partitioned VSAM linear data set. Consider the following for the SHCDS allocation:

- At a minimum, define and activate two SHCDSs and at least one spare SHCDS for recovery purposes to ensure duplexing of your data.

- Place the SHCDSs on volumes with global connectivity because VSAM RLS processing is only available on those systems that currently have access to the active SHCDS.
- The SHCDSs must not be shared outside the sysplex.
- Use SHAREOPTIONS(3,3).
- If SMS-managed, use a storage class with guaranteed space for the SHCDSs.
- Use the naming convention: SYS1.DFPSHCDS.*qualifier.Vvolser*, where:
  - *Qualifier* is a 1 to 8 character qualifier of your choice.
  - *Volser* is the volume serial number of the volume on which the data set resides.
- All SHCDSs should be of the same size.
- An SHCDS can have extents only on the same volume.

Both the primary and secondary SHCDS contain the same data. With the duplexing of the data, VSAM RLS ensures that processing can continue in case VSAM RLS loses connection to one SHCDS or the control data set got damaged. In that case, you can switch the spare SHCDS to active.

Attention: The SMSVSAM address space needs RACF UPDATE authority to SYS1.DFPSHCDS.

JCL to allocate the SHCDSs

You can use the sample JCL in Figure 7-19 to allocate your SHCDSs.

```
//STEP01 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DEFINE CLUSTER( -
        NAME(SYS1.DFPSHCDS.PRIMARY.VTOTSMA) -
        VOLUMES(TOTSMA) -
        MEGABYTES(10 10) -
        LINEAR -
        SHAREOPTIONS(3,3) -
        STORAGECLASS(GSPACE))
  DEFINE CLUSTER( -
        NAME(SYS1.DFPSHCDS.SECONDRY.VTOTCAT) -
        VOLUMES(TOTCAT) -
        MEGABYTES(10 10) -
        LINEAR -
        SHAREOPTIONS(3,3) -
        STORAGECLASS(GSPACE))
  DEFINE CLUSTER( -
        NAME(SYS1.DFPSHCDS.SPARE.VTOTSMS) -
        VOLUMES(TOTSMS) -
        MEGABYTES(10 10) -
        SHAREOPTIONS(3,3) -
        LINEAR -
        STORAGECLASS(GSPACE))
```

Figure 7-19 Allocating VSAM RLS SHCDSs

To calculate the size of the sharing control data sets, follow the guidelines in *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

Tip: Place the SHCDSs on different volumes to maximize availability. Avoid placing SHCDSs on volumes for which there might be extensive volume reserve activity.

SHCDS operations

Use the following command to activate your newly defined SHCDS for use by VSAM RLS:

- For the primary and secondary SHCDS, use

```
VARY SMS,SHCDS(SHCDS_name),NEW
```

- For the spare SHCDS use

```
VARY SMS,SHCDS(SHCDS_name),NEWSPARE
```

To display the used SHCDSs in an active configuration use:

```
D SMS,SHCDS
```

Figure 7-20 is an example of a SHCDS list in a sysplex.

```
D SMS,SHCDS
IEE932I 539
IGW612I 17:10:12    DISPLAY SMS,SHCDS
Name                      Size    %UTIL Status  Type
WTSCPLX2.VSBOX48        10800Kb     4%  GOOD    ACTIVE
WTSCPLX2.VSBOX52        10800Kb     4%  GOOD    ACTIVE
WTSCPLX2.VSBOX49        10800Kb     4%  GOOD    SPARE
-----------------           0Kb     0%  N/A     N/A
-----------------           0Kb     0%  N/A     N/A
-----------------           0Kb     0%  N/A     N/A
-----------------           0Kb     0%  N/A     N/A
-----------------           0Kb     0%  N/A     N/A
-----------------           0Kb     0%  N/A     N/A
-----------------           0Kb     0%  N/A     N/A
```

Figure 7-20 Example of SHCDS display

To delete logically either an active or a spare SHCDS, use:

```
VARY SMS,SHCDS(SHCDS_name),DELETE
```

Note: In the VARY SMS,SHCDS commands, the SHCDS name is not fully qualified. SMSVSAM takes as a default the first two qualifiers, which must always be SYS1.DFPSHCDS. You must specify only the last two qualifiers as the SHCDS names.

7.17 Update SMS configuration

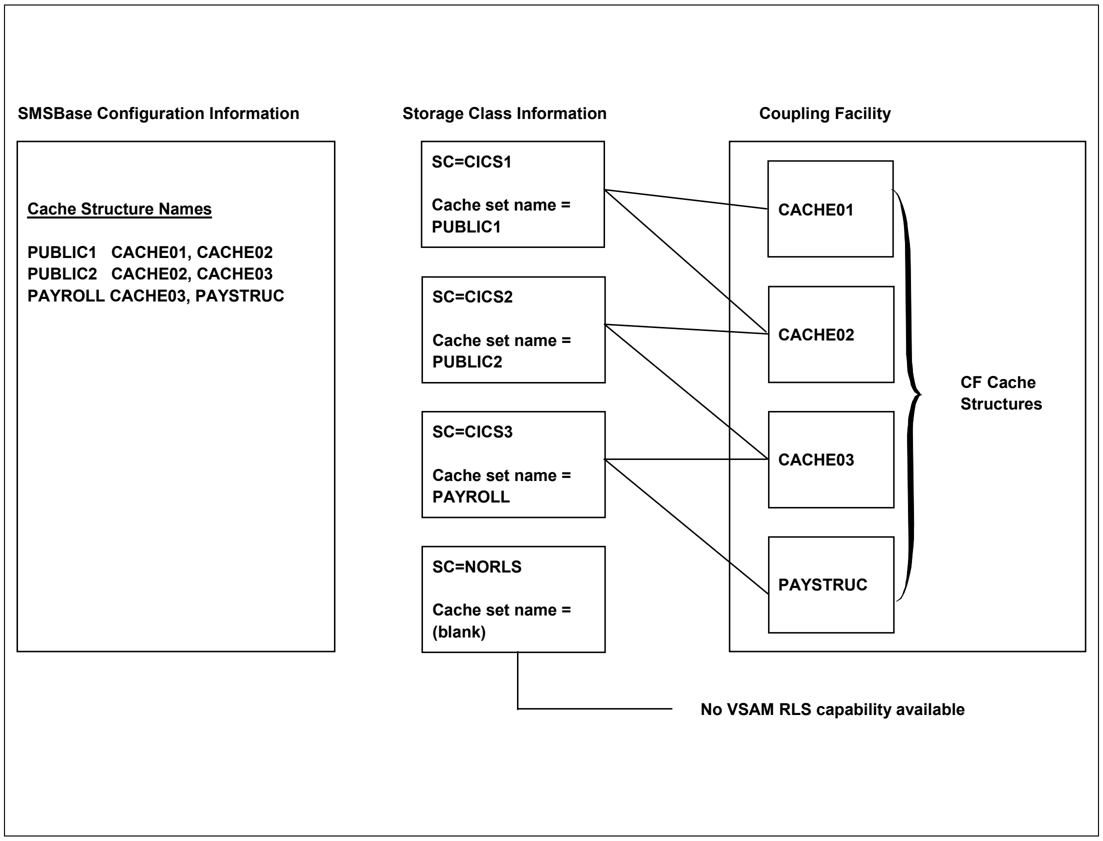


Figure 7-21 Example of SMS configuration with cache sets

Update the SMS configuration

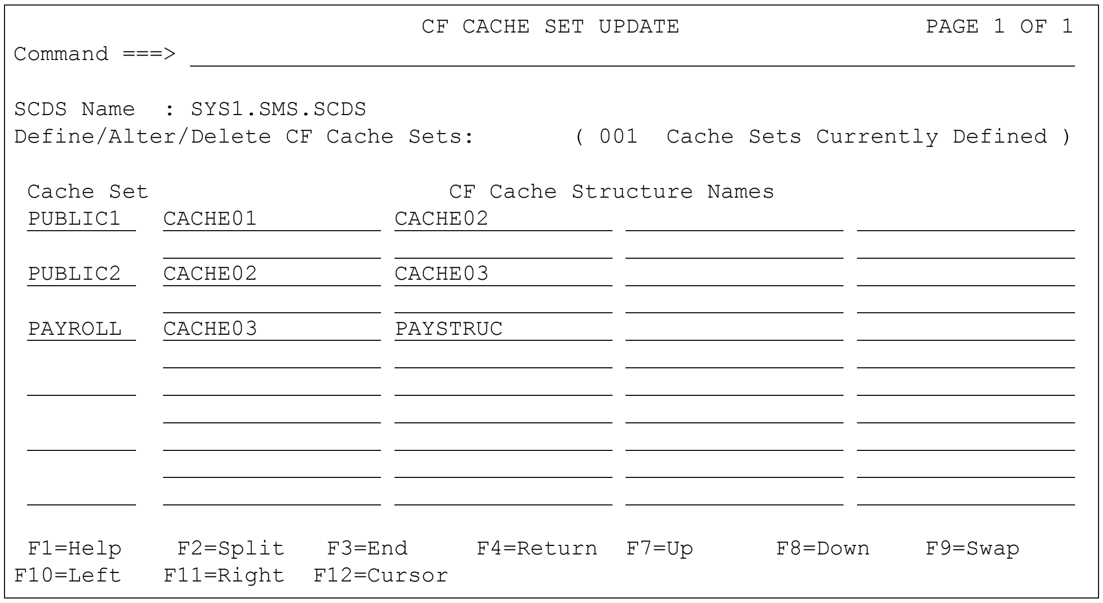
In order for DFSMSdfp to use the CF for VSAM RLS, after you define one or more CF cache structures to MVS, you must also add them in the SMS base configuration.

Define cache sets

Section 7.14, "Coupling facility structures for RLS sharing" on page 373 discusses defining CF cache structures. We now need to add the CF cache structures to the DFSMS base configuration. To do so, we need to associate them with a cache set name.

The following steps describe how to define a cache set and how to associate the cache structures to the cache set:

1. From the ISMF primary option menu for storage administrators, select option **8**, **Control Data Set.**
2. Select option **7**, **Cache Update,** and make sure that you specified the right SCDS name (SMS share control data set, do not mix up with SHCDS).
3. Define your CF cache sets (see Figure 7-22).



```
                               CF CACHE SET UPDATE                  PAGE 1 OF 1
Command ===> _________________________________________________________________

SCDS Name  : SYS1.SMS.SCDS
Define/Alter/Delete CF Cache Sets:       ( 001  Cache Sets Currently Defined )

 Cache Set                      CF Cache Structure Names
 PUBLIC1   CACHE01          CACHE02          ________________ ________________
           ________________ ________________ ________________ ________________
 PUBLIC2   CACHE02          CACHE03          ________________ ________________
           ________________ ________________ ________________ ________________
 PAYROLL   CACHE03          PAYSTRUC         ________________ ________________
           ________________ ________________ ________________ ________________
 ________  ________________ ________________ ________________ ________________
           ________________ ________________ ________________ ________________
 ________  ________________ ________________ ________________ ________________
           ________________ ________________ ________________ ________________
 ________  ________________ ________________ ________________ ________________

 F1=Help    F2=Split   F3=End     F4=Return  F7=Up      F8=Down    F9=Swap
F10=Left   F11=Right  F12=Cursor
```

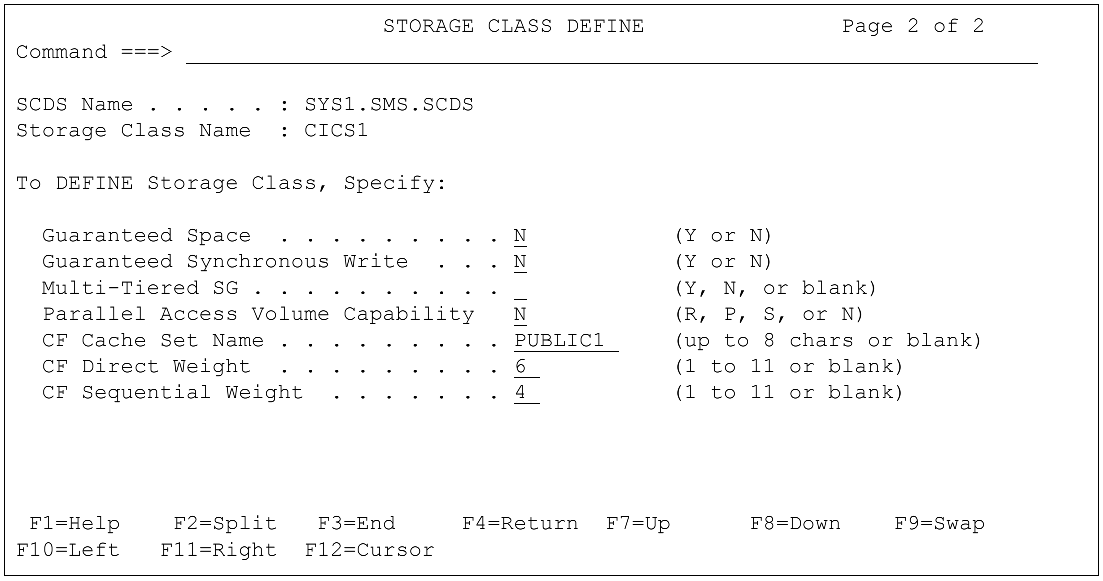
Figure 7-22 CF cache update panel in ISMF

SMS storage class changes

After you have defined your cache sets you need to assign them to one or more storage classes (SC), so that data sets associated with those SCs are eligible for VSAM RLS by using coupling facility cache structures.

Follow these steps to assign the CF cache sets:

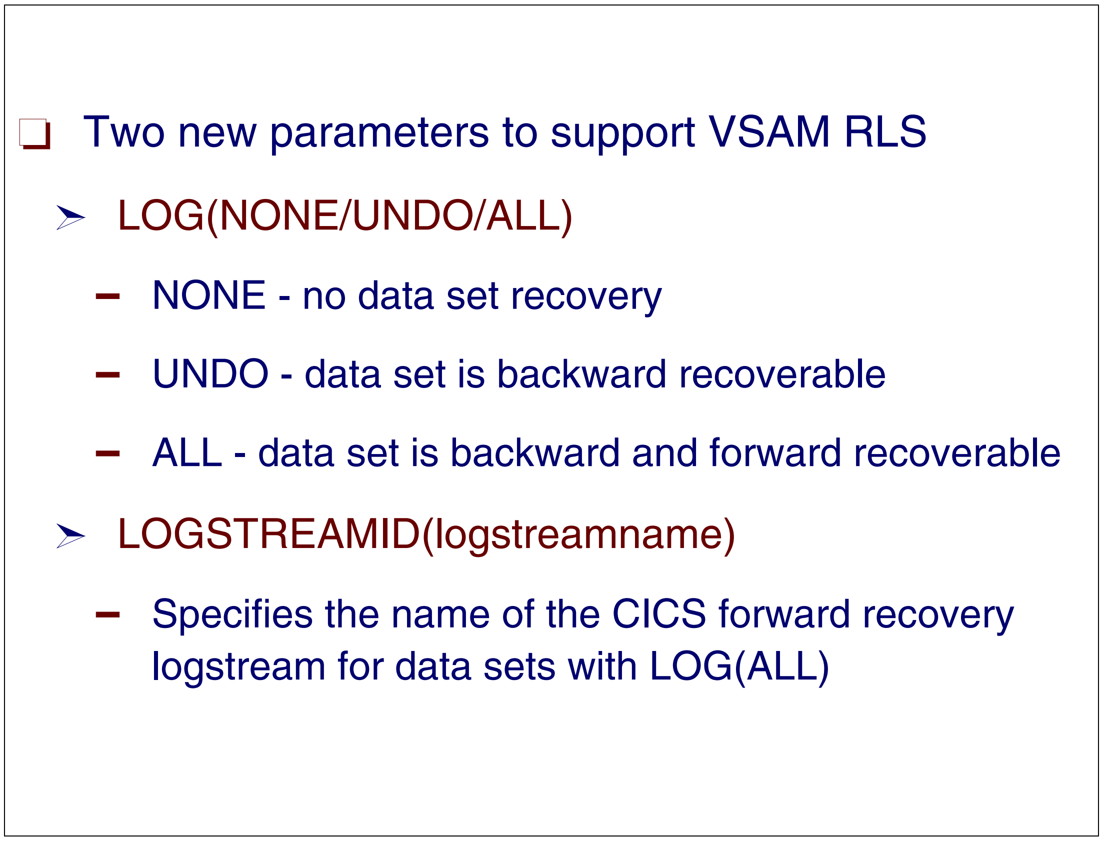
1. Select option **5**, **Storage Class**, from the ISMF primary option menu for storage administrators
2. Select option **3**, **Define**, and make sure, that you specified the right SCDS name
3. On the second page of the STORAGE CLASS DEFINE panel enter the name of the coupling facility cache set you defined in the base configuration (see Figure 7-23).
4. On the same panel enter values for direct and sequential weight. The higher the value, the more important it is that the data be assigned more cache resources.


```
                             STORAGE CLASS DEFINE               Page 2 of 2
Command ===> ___________________________________________________________________

SCDS Name . . . . . : SYS1.SMS.SCDS
Storage Class Name  : CICS1

To DEFINE Storage Class, Specify:

  Guaranteed Space  . . . . . . . . . N           (Y or N)
  Guaranteed Synchronous Write  . . . N           (Y or N)
  Multi-Tiered SG . . . . . . . . . . _           (Y, N, or blank)
  Parallel Access Volume Capability   N           (R, P, S, or N)
  CF Cache Set Name . . . . . . . . . PUBLIC1     (up to 8 chars or blank)
  CF Direct Weight  . . . . . . . . . 6           (1 to 11 or blank)
  CF Sequential Weight  . . . . . . . 4           (1 to 11 or blank)




 F1=Help    F2=Split   F3=End     F4=Return  F7=Up      F8=Down    F9=Swap
F10=Left   F11=Right  F12=Cursor
```
Figure 7-23 ISMF storage class definition panel, page 2

Note: Be sure to change your Storage Class ACS routines so that RLS data sets are assigned the appropriate storage class.

More detailed information about setting up SMS for VSAM RLS is in *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402.

7.18 Update data sets with log parameters



- Two new parameters to support VSAM RLS
  - LOG(NONE/UNDO/ALL)
    - NONE - no data set recovery
    - UNDO - data set is backward recoverable
    - ALL - data set is backward and forward recoverable
  - LOGSTREAMID(logstreamname)
    - Specifies the name of the CICS forward recovery logstream for data sets with LOG(ALL)

Figure 7-24 New parameters to support VSAM RLS

New parameters to support VSAM RLS

The two new parameters LOG and LOGSTREAMID are stored in the ICF catalog. You can use the IDCAMS DEFINE and ALTER commands to set the data set LOG attribute and to assign a log data set name.

Another way to assign the LOG attribute and a LOGSTREAMID is to use a data class that has those values already defined.

The LOG parameter is described in detail in "VSAM RLS/CICS data set recovery" on page 368.

Use the LOGSTREAMID parameter to assign a CICS forward recovery log stream to a data set which is forward recoverable.

JCL to define a cluster with LOG(ALL) and a LOGSTREAMID

The example in Figure 7-25 shows you how to define a VSAM data set that is eligible for RLS processing and that is forward recoverable.

```
//LABEL    JOB ...
//STEP1    EXEX PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DEFINE CLUSTER (NAME  (OTTI.IV4A62A1.RLS) -
                 CYL   (1  1)        -
                 RECSZ (5000 10009)  -
                 CISZ (20000)  -
                 IXD               -
                 KEYS (8 0)     -
                 FREESPACE(40 40) -
                 SHR (2,3)          -
                 REUSE              -
                 BWO(TYPECICS)     -
                 LOG(ALL)           -
                 LOGSTREAMID(CICS.IV4A62A1.DFHJ05) -
                 STORAGECLASS(CICS1) -
                )
/*
```

Figure 7-25 Sample JCL to define a recoverable VSAM data set RLS processing

For more information about the IDCAMS DEFINE and ALTER commands, see *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

JCL to define a log stream

You can use the JCL in Figure 7-26 to define a log stream.

```
//LABEL    JOB ...
//STEP010  EXEC PGM=IXCMIAPU
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
    DELETE LOGSTREAM NAME(CICS.IV4A62A1.DFHJ05)
    DATA TYPE(LOGR) REPORT(NO)
    DEFINE LOGSTREAM
           NAME(CICS.IV4A62A1.DFHJ05)
           DASDONLY(YES)
           STG_SIZE(50)
           LS_SIZE(25)
           HIGHOFFLOAD(80)
           LOWOFFLOAD(0)
/*
```

Figure 7-26 JCL to define a log stream

For information about the IXCMIAPU utility, see *z/OS MVS Setting Up a Sysplex,* SA22-7625.

7.19 The SMSVSAM address space

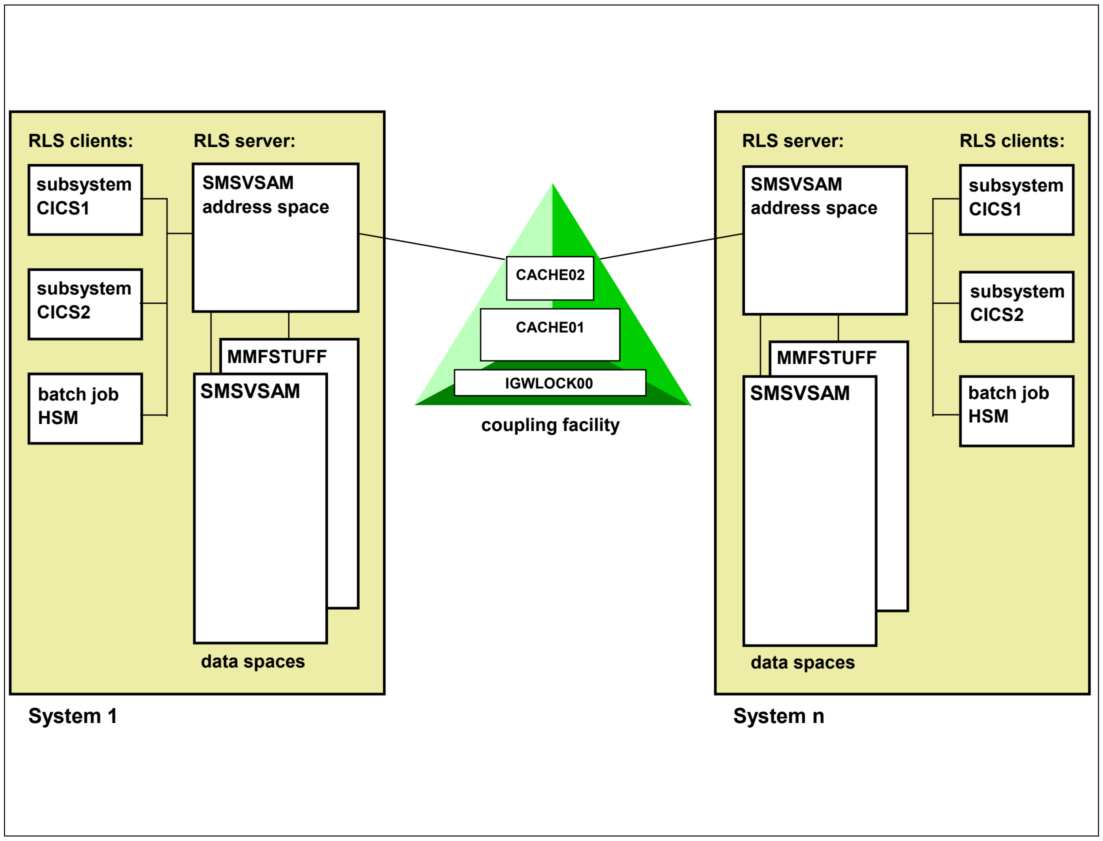


Figure 7-27 SMSVSAM address space

The SMSVSAM address space

SMSVSAM is the MVS job name of the VSAM RLS address space. It is started automatically at IPL time if RLSINIT(YES) is specified in the IGDSMSxx member of SYS1.PARMLIB. Another way to start it is by operator command (see "Interacting with VSAM RLS" on page 388).

The SMSVSAM address space needs to be started on each system where you want to exploit VSAM RLS. It is responsible for centralizing all processing necessary for cross-system sharing, which includes one connect per system to XCF lock, cache, and VSAM control block structures.

The SMSVSAM address space owns two data spaces:

- SMSVSAM

  Contains some VSAM RLS control blocks and a system-wide buffer pool

- MMFSTUFF

  Collect activity monitoring information that is used to produce SMF records

Terminology

We use the following terms to describe an RLS environment:

- RLS server

  The SMSVSAM address space is also referred to as the RLS server.

- RLS client

  Any address space that invokes an RLS function that results in a program call to the SMSVSAM address space is called an RLS client. Those address spaces can be CICS regions as well as batch jobs.

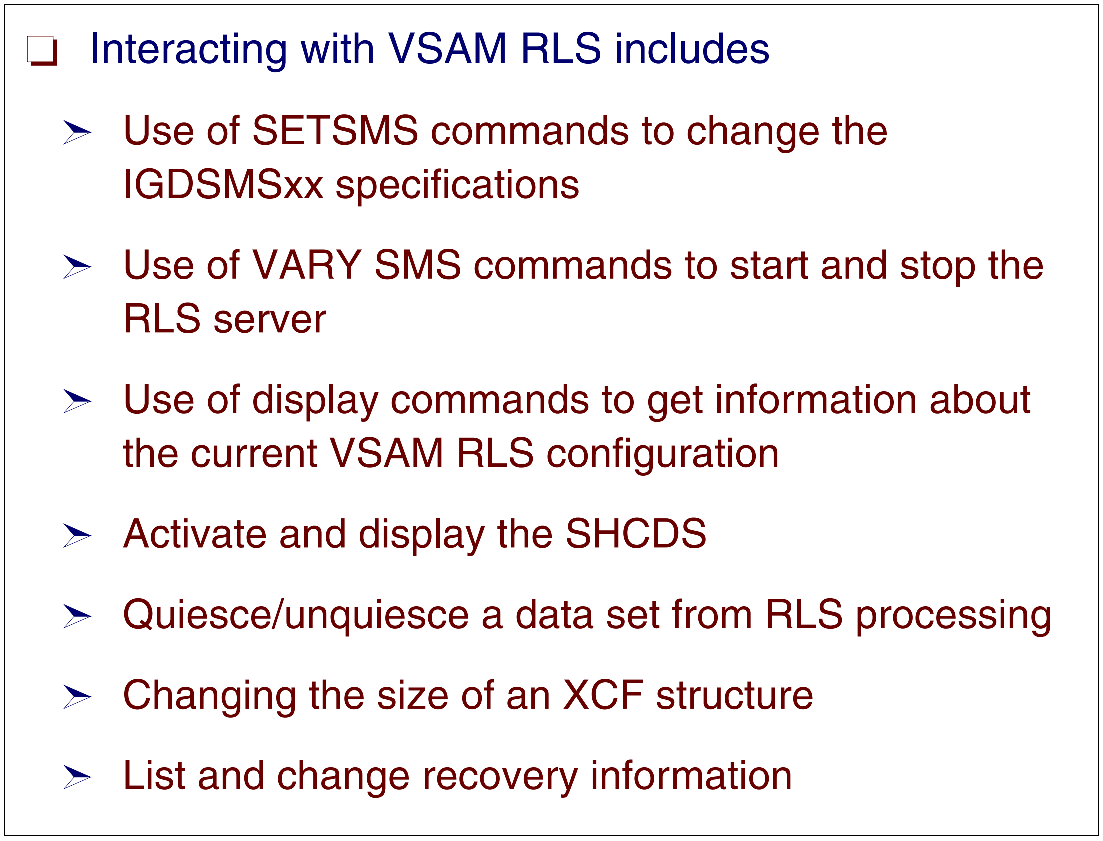
- Recoverable subsystem

  A subsystem is an RLS client space that *registers* with the SMSVSAM address space as an address space that will provide transactional and data set recovery. CICS, for example, is a *recoverable subsystem*.

- Batch job

  An RLS client space that does *not* first register with SMSVSAM as a recoverable subsystem is called a batch job. An example for such a batch job is HSM.

7.20 Interacting with VSAM RLS



❑ Interacting with VSAM RLS includes

- Use of SETSMS commands to change the IGDSMSxx specifications
- Use of VARY SMS commands to start and stop the RLS server
- Use of display commands to get information about the current VSAM RLS configuration
- Activate and display the SHCDS
- Quiesce/unquiesce a data set from RLS processing
- Changing the size of an XCF structure
- List and change recovery information

Figure 7-28 Interacting with VSAM RLS

Interacting with VSAM RLS

This section is a short overview of some important commands you need to know to monitor and control the VSAM RLS environment.

SETSMS command

Use the SETSMS command to overwrite the PARMLIB specifications for IGDSMSxx. The syntax is:

```
SETSMS CF-TIME(nnn|3600)
       DEADLOCK_DETECTION(iiii,kkkk)
       RLSINIT
       RLS_MAXCFFEATURELEVEL({A|Z})
       RLS_MAX_POOL_SIZE(nnnn|100)
       SMF_TIME(YES|NO)
```

For information about these PARMLIB values refer to 7.15, “Update PARMLIB with VSAM RLS parameters” on page 376.

VARY SMS command

Use the VARY SMS command to control SMSVSAM processing.

- Start the VSAM RLS server address space on a single system

If not started during IPL because of RLSINIT(NO) in the IGDSMSxx PARMLIB member, you can start the RLS server manually with:

```
V SMS,SMSVSAM,ACTIVE
```

- Stop the RLS server address space

  If the SMSVSAM address space fails, it is automatically restarted. If you want to stop the SMSVSAM address space permanently on a single system use:

  ```
  V SMS,SMSVSAM,TERMINATESERVER
  ```

  Use this command only for specific recovery scenarios that require the SMSVSAM server to be down and to not restart automatically.

- Fallback from RLS processing

  A detailed procedure about falling back from RLS processing is described in *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402. You should read it before using the following command:

  ```
  V SMS,SMSVSAM,FALLBACK
  ```

  This command is used as the last step in the disablement procedure to fall back from SMSVSAM processing.

- Further VARY SMS commands

  There are additional VARY SMS commands available to interact with VSAM RLS. They are:

  ```
  V SMS,CFCACHE(cachename),ENABLE|QUIESCE
        CFVOL(volid),ENABLE|QUIESCE
        MONDS(dsname[,dsname...]),ON|OFF
        SHCDS(shcdsname),NEW|NEWSPARE|DELETE
        SMSVSAM,SPHERE(spherename),ENABLE
                FORCEDELETELOCKSTRUCTURE
  ```

  Refer to *z/OS MVS System Commands,* SA22-7627 for information about these commands.

Display commands

There are a several display commands available to provide RLS-related information.

- Display the status of the SMSVSAM address space:

  ```
  DISPLAY SMS,SMSVSAM{,ALL}
  ```

  Specify ALL to see the status of all the SMSVSAM servers in the sysplex.

- Display information about the coupling facility *cache* structure:

  ```
  DISPLAY SMS,CFCACHE(CF_cache_structure_name|*)
  ```

- Display information about the coupling facility *lock* structure IGWLOCK00:

  ```
  DISPLAY SMS,CFLS
  ```

  This information includes the lock rate, lock contention rate, false contention rate, and average number of requests waiting for locks.

- Display XCF information for a CF structure:

  ```
  DISPLAY XCF,STR,STRNAME=structurename
  ```

  Provides information like status, type, and policy size for a CF structure.

For further DISPLAY commands refer to *z/OS MVS System Commands,* SA22-7627.

Activate and display the sharing control data sets

Refer to 7.16, “Define sharing control data sets” on page 378 for information about how to activate and display the VSAM RLS sharing control data sets.

Quiesce and unquiesce a data set from RLS processing

In 7.12, “The batch window problem” on page 371 we discussed the reasons for quiescing data sets from RLS processing.

There are different ways to quiesce/unquiesced a data set.

- Use the CICS command:

  ```
  CEMT SET DSNAME(dsname) QUIESCED|UNQUIESCED
  ```

- Use an equivalent SPI command in a user program.
- Use the system command:

  ```
  VARY SMS,SMSVSAM,SPHERE(dsname),QUIESCE|ENABLE
  ```

The quiesce status of a data set is set in the catalog and is shown in an IDCAMS LISTCAT output for the data set. See “Interpreting RLSDATA in an IDCAMS LISTCAT output” on page 393 for information about interpreting LISTCAT outputs.

Changing the size of an XCF structure

Use the following command to change the size of a coupling facility structure:

```
SETXCF START,ALTER,STRNAME=CF_cachestructurename,SIZE=newsize
```

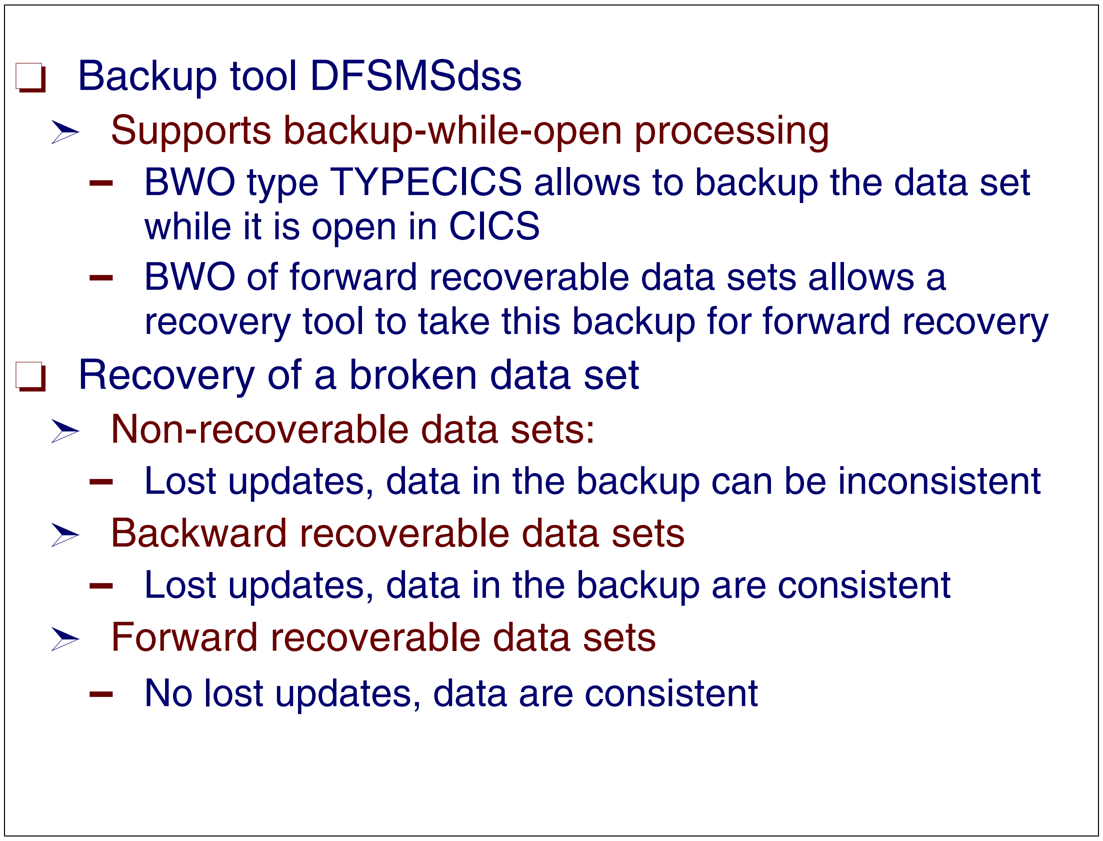
This new size can be larger or smaller than the size of the current CF cache structure, but it cannot be larger than the maximum size specified in the CFRM policy. The `SETXCF START,ALTER` command will not work unless the structure’s ALLOW ALTER indicator is set to YES.

List and change recovery information

You can use the IDCAMS command SHCDS to list and change recovery information kept by the SMSVSAM server. It also resets VSAM record-level sharing settings in catalogs, allowing for non-RLS access or fallback from RLS processing. For the single parameters for this command refer to *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

Attention: This section provides you only an overview about useful commands you should know to work with VSAM RLS. Before you use any of these commands other than the **`DISPLAY`** command, read the official z/OS manuals carefully.

7.21 Backup and recovery of CICS VSAM data sets



- Backup tool DFSMSdss
  - Supports backup-while-open processing
    - BWO type TYPECICS allows to backup the data set while it is open in CICS
    - BWO of forward recoverable data sets allows a recovery tool to take this backup for forward recovery
- Recovery of a broken data set
  - Non-recoverable data sets:
    - Lost updates, data in the backup can be inconsistent
  - Backward recoverable data sets
    - Lost updates, data in the backup are consistent
  - Forward recoverable data sets
    - No lost updates, data are consistent

Figure 7-29 Backup and recovery of CICS VSAM data sets

Backup tool DFSMSdss

DFSMSdss supports *backup-while-open* (BWO) serialization, which can perform backups of data sets that are open for update for long periods of time. It can also perform a logical data set dump of these data sets even if another application has them serialized. Backup-while-open is a better method than using SHARE or TOLERATE(ENQFAILURE) for dumping CICS VSAM file-control data sets that are in use and open for update.

When you dump data sets that are designated by CICS as eligible for backup-while-open processing, data integrity is maintained through serialization interactions between:

- CICS (database control program)
- VSAM RLS
- VSAM record management
- DFSMSdfp
- DFSMSdss

Backup-while-open

In order to allow DFSMSdss to take a backup while your data set is open by CICS, you need to define the data set with the BWO attribute *TYPECICS* or assign a data class with this attribute.

- TYPECICS

  Use TYPECICS to specify BWO in a CICS environment. For RLS processing, this activates BWO processing for CICS. For non-RLS processing, CICS determines whether to use this specification or the specification in the CICS FCT. The BWO type is stored in the ICF catalog.

Backup-while-open of a forward recoverable data set

If you use the DSS BWO processing for a forward recoverable data set, CICS will log the start and end of the copy/backup operation. The data set can then be fully recovered from this backup.

For information about the BWO processing refer to *z/OS DFSMSdss Storage Administration Reference,* SC35-0424.

Recover a CICS VSAM data set

Sometimes it is necessary to recover a data set, for example if it got broken.

- Recovery of a non-recoverable data set

  Data sets with LOG(NONE) are considered non-recoverable. To recover such a data set, restore the last backup of the data set. All updates to this data set after the backup was taken are lost. If the backup was taken after a transaction failed (did not commit), the data in the backup might be inconsistent.

- Recovery of a backward recoverable data set

  Data sets with the LOG(NONE) attribute are considered backward recoverable. To recover such a data set, restore the last backup of the data set. All updates to this data set after the backup was taken are lost. The data in the backup are consistent

- Recovery of a forward recoverable data set

  Data sets with LOG(ALL) and a log stream assigned are forward recoverable. Restore the last backup of the data set. Then run a tool like *CICS VSAM Recovery* (CICSVR) which uses the forward recovery log to redrive all committed updates until the data set got broken. No updates are lost.

7.22 Interpreting RLSDATA in an IDCAMS LISTCAT output

```
CLUSTER ------- OTTI.IV4A62A1.RLS
     IN-CAT --- CATALOG.MVSICFU.VO260C1
     HISTORY
       DATASET-OWNER-----(NULL) CREATION--------2005.122
       RELEASE----------------2 EXPIRATION------0000.000
     SMSDATA
       STORAGECLASS ------CICS1 MANAGEMENTCLASS-CICSRLSM
       DATACLASS --------(NULL) LBACKUP ---0000.000.0000
       BWO STATUS------00000000 BWO TIMESTAMP---00000 00:00:00.0
       BWO-------------TYPECICS
     RLSDATA
       LOG ------------------ALL RECOVERY REQUIRED --(NO)
       VSAM QUIESCED -------(NO) RLS IN USE ---------(YES)
       LOGSTREAMID--------------CICS.IV4A62A1.DFHJ05
       RECOVERY TIMESTAMP LOCAL-----X'0000000000000000'
       RECOVERY TIMESTAMP GMT-------X'0000000000000000'
```

Figure 7-30 Sample RLSDATA in an IDCAMS LISTCAT output

RLSDATA in an IDCAMS LISTCAT output

The RLSDATA in the output of an IDCAMS LISTCAT job shows you the RLS status of the data set and recovery information.

You can use the sample JCL in Figure 7-31 to run an IDCAMS LISTCAT job.

```
//LABEL   JOB ...
//S1      EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
  LISTC ENT(OTTI.IV4A62A1.RLS) ALL
/*
```

Figure 7-31 Sample JCL for IDCAMS LISTCAT

RLSDATA

RLSDATA contains the following information:

- LOG

  This field shows you the type of logging used for this data set. It can be NONE, UNDO or ALL.

- RECOVERY REQUIRED

  This field indicates whether the sphere is currently in the process of being forward recovered.

- VSAM QUIESCED

  If this value is YES, the data set is currently quiesced from RLS processing so no RLS access is possible until it is unquiesced (see also “Interacting with VSAM RLS” on page 388).

- RLS IN USE

  If this value is YES it means:

  - The data set was last opened for RLS access.
  - The data set is not opened for RLS processing but is recoverable and either has retained locks protecting updates or is in a lost locks state.

  Note: If the RLS-IN-USE indicator is on, it doesn’t mean that the data set is currently in use by VSAM RLS. It just means that the last successful open was for RLS processing. Non-RLS open will always attempt to call VSAM RLS if the RLS-IN-USE bit is on in the catalog. This bit is a safety net to prevent non-RLS users from accessing a data set which may have retained or lost locks associated with it. The RLS-IN-USE bit is set on by RLS open and is left on after close. This bit is only turned off by a successful non-RLS open or by the IDCAMS `SHCDS CFRESET` command.

- LOGSTREAMID

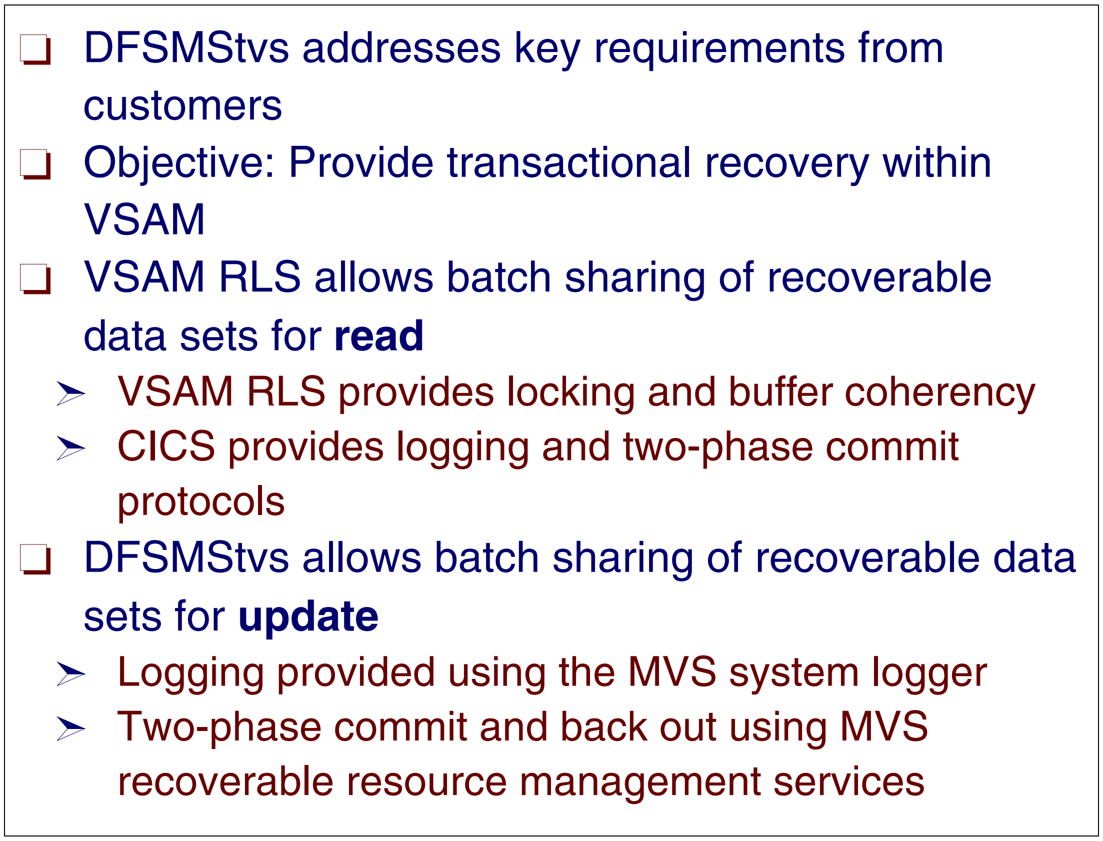
  This value tells you the forward recovery log stream name for this data set if the LOG attribute has the value of ALL.

- RECOVERY TIMESTAMP

  The recovery time stamp gives the time the most recent backup was taken when the data set was accessed by CICS using VSAM RLS.

All LISTCAT keywords are described in Appendix B of *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

7.23 DFSMStvs introduction



- ❑ DFSMStvs addresses key requirements from customers
- ❑ Objective: Provide transactional recovery within VSAM
- ❑ VSAM RLS allows batch sharing of recoverable data sets for **read**
  - ➢ VSAM RLS provides locking and buffer coherency
  - ➢ CICS provides logging and two-phase commit protocols
- ❑ DFSMStvs allows batch sharing of recoverable data sets for **update**
  - ➢ Logging provided using the MVS system logger
  - ➢ Two-phase commit and back out using MVS recoverable resource management services

Figure 7-32 DFSMS Transactional VSAM Services (DFSMStvs) introduction

Why DFSMS Transactional VSAM Services (DFSMStvs)

There following key requirements were expressed by customers running CICS applications in RLS mode:

- Address the *batch window problem* for CICS and batch sharing of VSAM data sets.

  In "The batch window problem" on page 371 we discussed the problem with the batch window. Customers reported batch windows ranging from two to ten hours. The programs run during the batch window consisted of both in-house applications and vendor-written applications.
- Allow batch *update* sharing concurrent with CICS use of recoverable data.
- Allow *multiple* batch update programs to run concurrently.
- Allow programs to interact with multiple resource managers such as IMS and DB2.
- Allow full transactional read/write access from new Java programs to existing VSAM data.

Objective of DFSMStvs

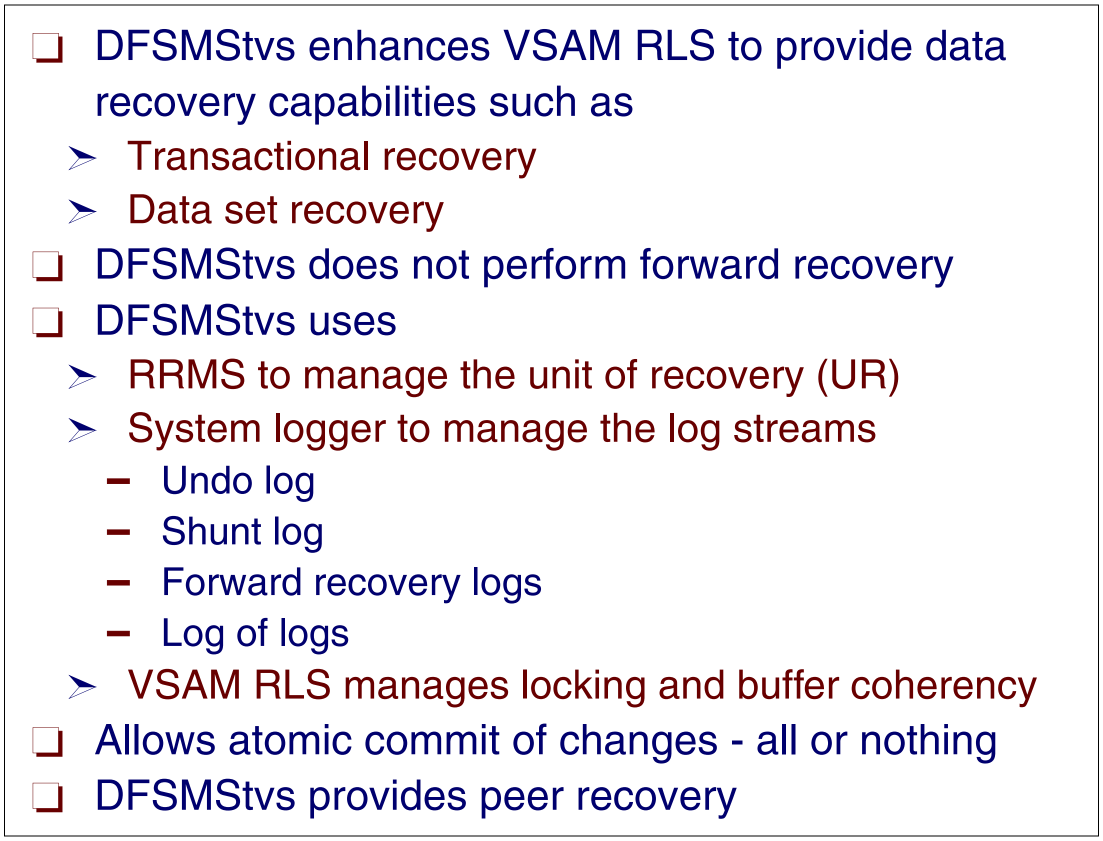
The objective of DFSMStvs is to provide transactional recovery *directly* within VSAM. It is an extension to VSAM RLS. It allows *any* job or application that is designed for data sharing to read/write share VSAM recoverable files.

DFSMStvs is a follow-on project/capability based on VSAM RLS. VSAM RLS supports CICS as a transaction manager. This provides sysplex data sharing of VSAM recoverable files when accessed through CICS. CICS provides the necessary unit-of-work management, undo/redo logging, and commit/back out functions. VSAM RLS provides the underlying sysplex-scope locking and data access integrity.

DFSMStvs adds logging and commit/back out support to VSAM RLS. DFSMStvs requires and supports the RRMS (recoverable resource management services) component as the commit or sync point manager.

DFSMStvs provides a level of data sharing with built-in transactional recovery for VSAM recoverable files that is comparable with the data sharing and transactional recovery support for data bases provided by DB2 and IMSDB.

7.24 Overview of DFSMStvs



- DFSMStvs enhances VSAM RLS to provide data recovery capabilities such as
  - Transactional recovery
  - Data set recovery
- DFSMStvs does not perform forward recovery
- DFSMStvs uses
  - RRMS to manage the unit of recovery (UR)
  - System logger to manage the log streams
    - Undo log
    - Shunt log
    - Forward recovery logs
    - Log of logs
  - VSAM RLS manages locking and buffer coherency
- Allows atomic commit of changes - all or nothing
- DFSMStvs provides peer recovery

Figure 7-33 DFSMStvs overview

Enhancements of VSAM RLS

DFSMStvs enhances VSAM RLS to perform data recovery in the form of:

- Transactional recovery (see "Transactional recovery" on page 370)
- Data set recovery (see "VSAM RLS/CICS data set recovery" on page 368)

Before DFSMStvs, those two types of recovery were only supported by CICS.

CICS performs the *transactional recovery* for data sets defined with a LOG parameter UNDO or ALL.

For forward recoverable data sets (LOG(ALL)) CICS also records updates in a log stream for forward recovery. CICS itself does not perform forward recovery, it performs only logging. For forward recovery you need a utility like CICS VSAM recovery (CICSVR).

Like CICS, DFSMStvs also provides transactional recovery and logging.

Without DFSMStvs, batch jobs cannot perform transactional recovery and logging. That is the reason batch jobs were granted only *read* access to a data set that was opened by CICS in RLS mode. A batch window was necessary to run batch updates for CICS VSAM data sets.

With DFSMStvs, batch jobs can perform transactional recovery and logging concurrently with CICS processing. Batch jobs can now *update* data sets while they are in use by CICS. No batch window is necessary any more.

Like CICS, DFSMStvs does not perform data set forward recovery.

Components used by DFSMStvs

The following components are involved in DFSMStvs processing:

- RRMS to manage the unit of recovery

  DFSMStvs uses the recoverable resource management services (RRMS) to manage the unit of recovery that is needed for transactional recovery. More information about RRMS is in the next section.

- System logger to manage log streams

  There are three kinds of logs used by DFSMStvs:

  - Undo log: Used for transactional recovery (back out processing)
  - Forward recovery log: Used to log updates against a data set for forward recovery
  - Shunt log: Used for long running or failed units of recovery

  You can also define a log of logs for use to automate forward recovery.

  These logs are maintained by the MVS system logger. For information about the different log types see “DFSMStvs logging” on page 403

- VSAM RLS - used for record locking and buffering

  We already discussed the need for VSAM RLS in the previous sections.

Atomic commit of changes

DFSMStvs allows atomic commit of changes. That means, if multiple updates to different records are necessary to complete a transaction, either all updates are committed or none are, if the transaction does not complete. See also “Transactional recovery” on page 370 and “Atomic updates” on page 401.

Peer recovery

Peer recovery allows DFSMStvs to recover for a failed DFSMStvs instance to clean up any work that was left in an incomplete state and to clear retained locks that resulted from the failure.

For more information about peer recovery refer to *z/OS DFSMStvs Planning and Operation Guide,* SC26-7348.

7.25 DFSMStvs use of z/OS RRMS

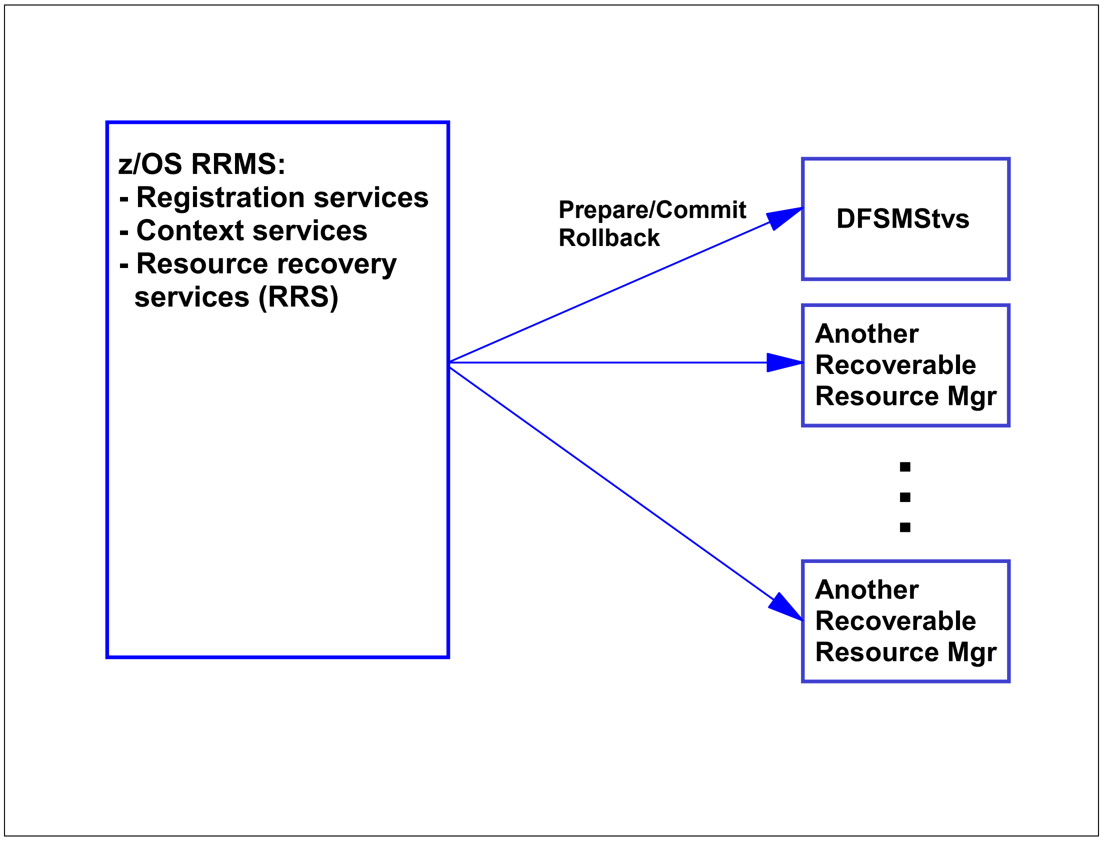

z/OS RRMS:
- Registration services
- Context services
- Resource recovery services (RRS)
Prepare/Commit
Rollback
DFSMStvs
Another Recoverable Resource Mgr
Another Recoverable Resource Mgr

Figure 7-34 DFSMStvs and RRMS

The recoverable resource management services (RRMS)

z/OS provides recoverable resource management services (RRMS) comprising:

- Registration services
- Context services
- Resource recovery services (RRS), which acts as syncpoint manager

The role of resource recovery services (RRS)

RRS provides the *syncpoint* services and is the most important component from a DFSMStvs use perspective.

DFSMStvs is a *recoverable resource manager*. It is *not* a commit or sync point manager. DFSMStvs interfaces with the z/OS syncpoint manager (RRS).

When an application issues a commit request directly to z/OS or indirectly through a sync point manager that interfaces with the z/OS syncpoint manager, DFSMStvs is invoked to participate in the *2-phase commit process*.

Other resource managers (like DB2) whose recoverable resources were modified by the transaction are also invoked by the z/OS syncpoint manager, thus providing a commit scope across the multiple resource managers.

RRS is a system-wide *commit coordinator*. It enables transactions to update protected resources managed by many resource managers.

It is RRS that provides the means to implement two-phase commit, but a resource manager must also use registration services and context services in conjunction with resource recovery services.

Two-phase commit

The *two-phase commit* protocol is a set of actions used to make sure that an application program either makes *all* changes to the resources represented by a single unit of recovery (UR), or makes *no* changes at all. This protocol verifies that either all changes or no changes are applied even if one of the elements (such as the application, the system, or the resource manager) fails. The protocol allows for restart and recovery processing to take place after system or subsystem failure.

For a discussion of the term *unit of recovery* see “Unit of work and unit of recovery” on page 402.

7.26 Atomic updates

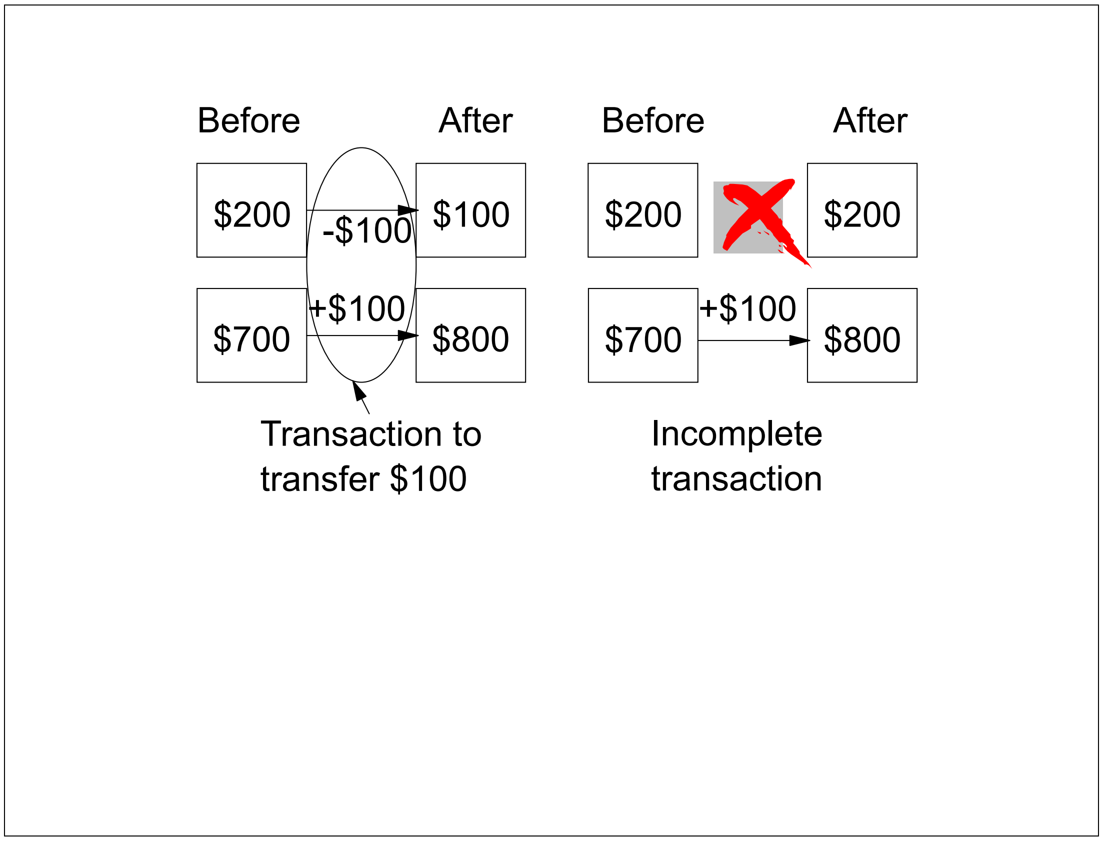


Figure 7-35 Example of an atomic update

Atomic updates

A transaction is known as *atomic* when an application changes data in multiple resource managers as a single transaction, and all of those changes are accomplished through a single commit request by a sync point manager. If the transaction is successful, all the changes are committed. If any piece of the transaction is not successful, then all changes are backed out. An *atomic instant* occurs when the sync point manager in a two-phase commit process logs a commit record for the transaction.

Refer also to “Transactional recovery” on page 370 for information about recovery of an uncompleted transaction.

7.27 Unit of work and unit of recovery

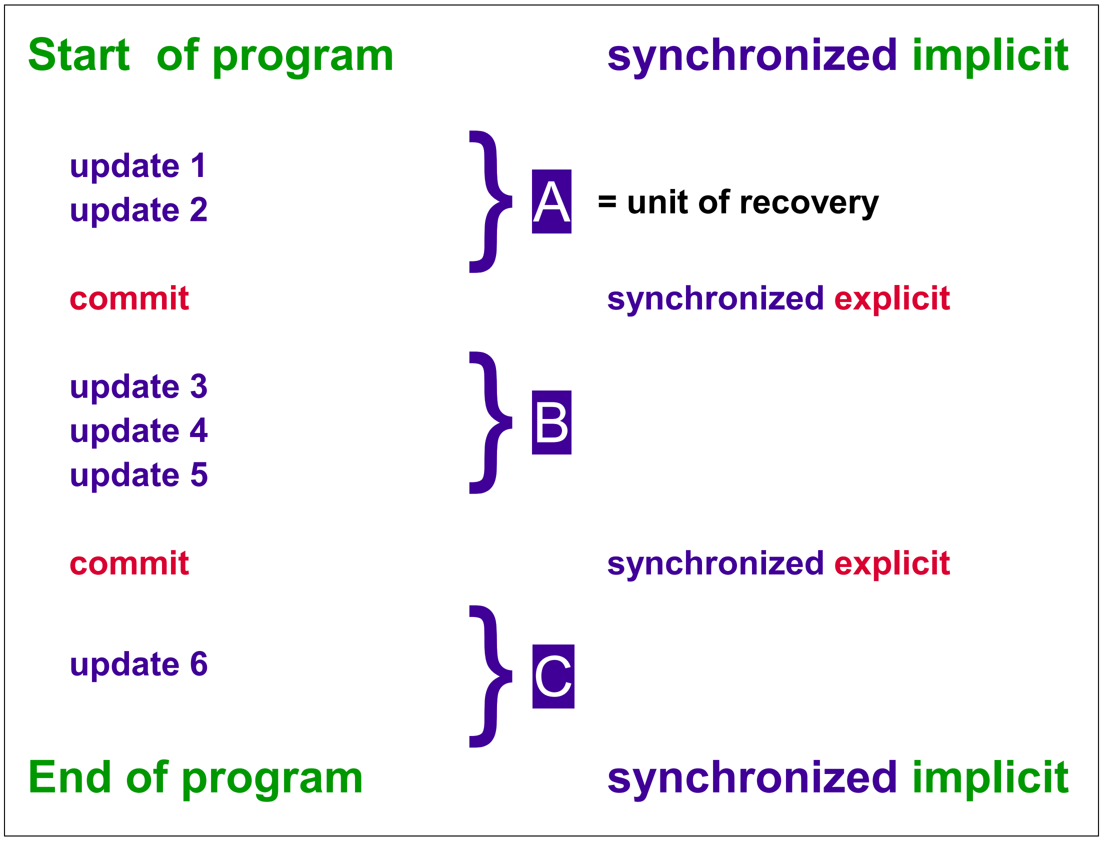


Figure 7-36 Unit of recovery example

Unit of work and unit of recovery

A *unit of work* (UOW) is the term used in CICS publications for a set of updates that are treated as an atomic set of changes.

RRS uses *unit of recovery* (UR) to mean much the same thing. So, a unit of recovery is the *set of updates between synchronization points*. There are implicit synchronization points at the start and at the end of a transaction. There should also be explicit synchronization points requested by an application within a transaction or batch job. It is preferable to use explicit synchronization for greater control of the number of updates in a unit of recovery.

Changes to data are durable after a synchronization point. That means that the changes survive any subsequent failure.

In Figure 7-36 there are three units of recovery, noted as A, B and C. The synchronization points between the units of recovery are either:

- *Implicit* - At the start and end of the program
- *Explicit* - When requested by commit

7.28 DFSMStvs logging

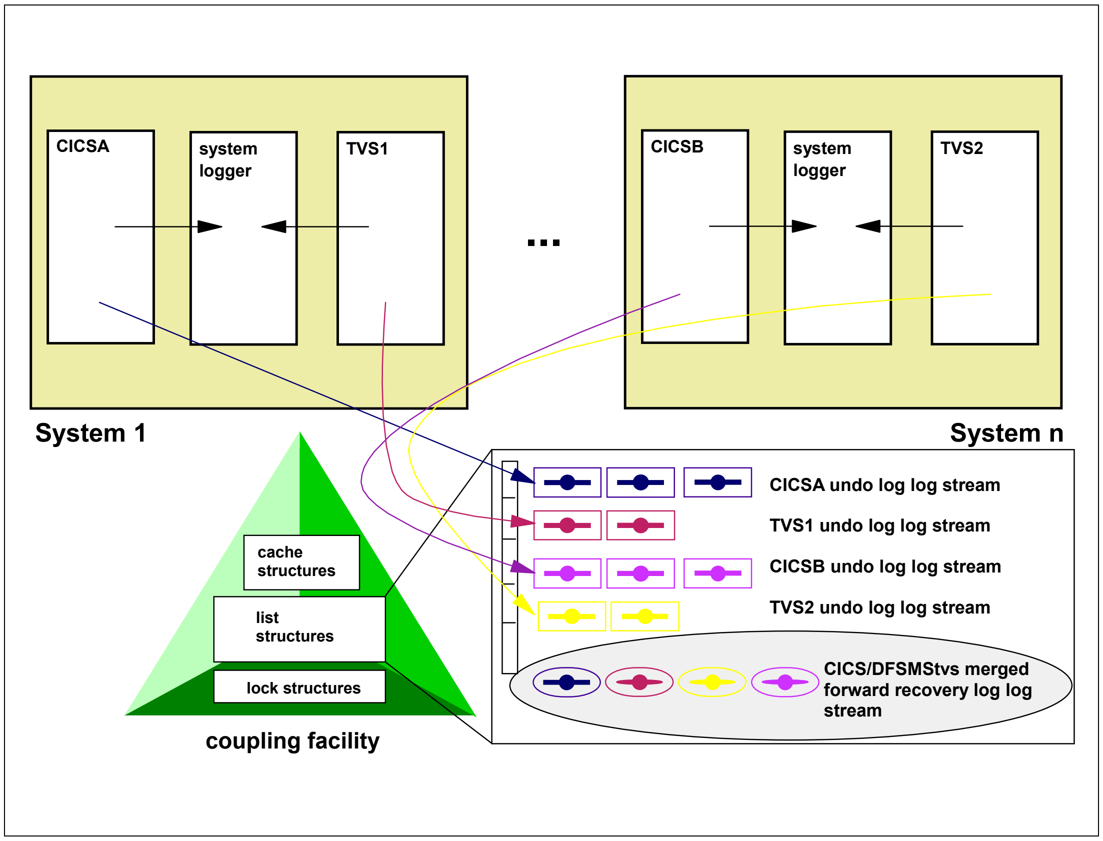


Figure 7-37 DFSMStvs logging

DFSMStvs logging

DFSMStvs logging uses the z/OS system logger. The design of DFSMStvs logging is similar to the design of CICS logging. Forward recovery logstreams for VSAM recoverable files will be shared across CICS *and* DFSMStvs. CICS will log changes made by CICS transactions while DFSMStvs will log changes made by its callers.

The system logger

The system logger (IXGLOGR) is an MVS component that provides a rich set of services that allow another component or an application to write, browse, and delete log data. The system logger is used because it can merge log entries from many z/OS images to a single log stream, where a log stream is simply a set of log entries.

Types of logs

There are different types of logs involved in DFSMStvs (and CICS) logging. They are:

- Undo logs (mandatory, one per image) - tvsname.IGWLOG.SYSLOG

  The backout or undo log contains images of changed records for recoverable data sets as they existed prior to being changed. It is used for *transactional recovery* to back out uncommitted changes if a transaction failed.

- Shunt logs (mandatory, one per image) - tvsname.IGWSHUNT.SHUNTLOG

  The shunt log is used when backout requests fail and for long running units of recovery.

- Log of logs (optional, shared with CICS and CICSVR) - Default name is CICSUSER.CICSVR.DFHLGLOG

  The log of logs contains copies of log records that are used to automate forward recovery.

- Forward recovery logs (optional, shared with CICS) - Name of your choice

  The forward recovery log is used for data sets defined with LOG(ALL). It is used for *forward recovery*. It needs to be defined in the LOGSTREAMID parameter of the data set. For more information see “VSAM RLS/CICS data set recovery” on page 368.

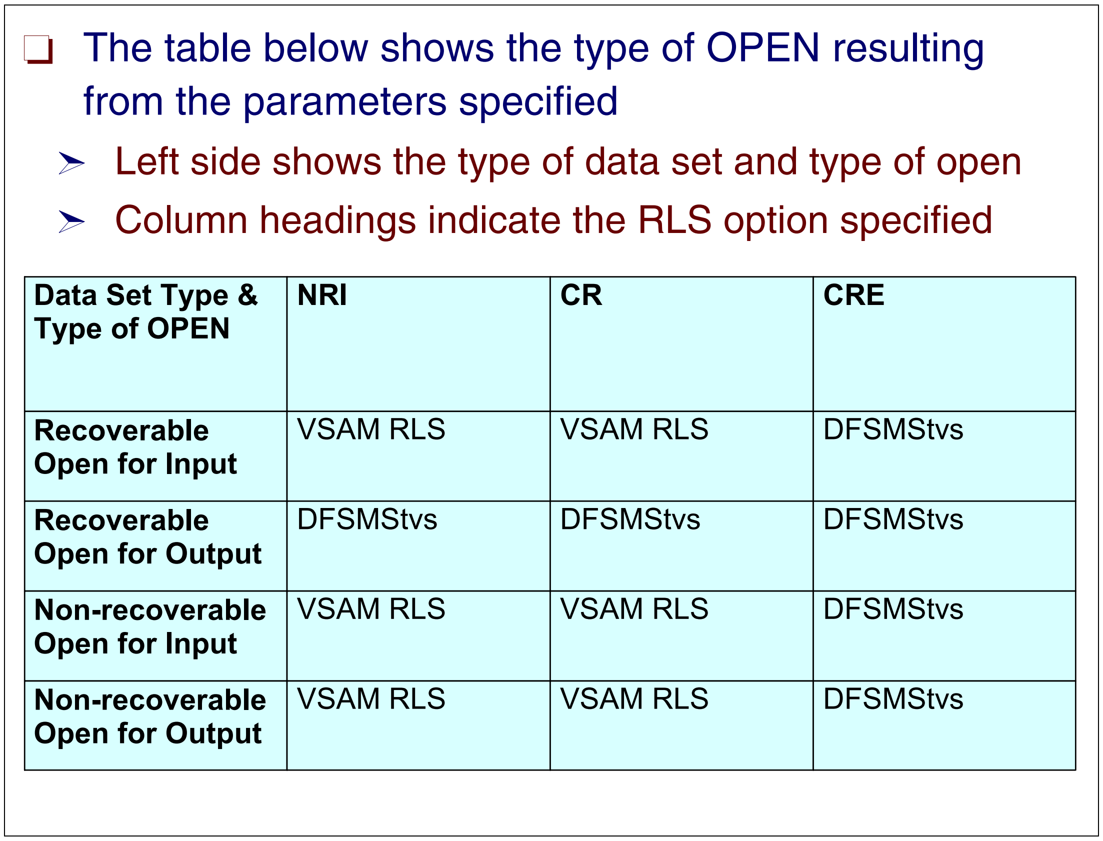
The system logger writes log data to log streams. The log streams are put in list structures in the coupling facility (except for DASDONLY log streams).

As Figure 7-37 on page 403 shows, you can merge forward recovery logs for use by CICS and DFSMStvs. You can also share it by multiple VSAM data sets. You cannot share an undo log by CICS and DFSMStvs; you need one per image.

For information about how to define log streams and list structures refer to 7.32, “Prepare for logging” on page 409.

DFSMStvs needs UPDATE RACF access to the log streams.

7.29 Accessing a data set with DFSMStvs



- The table below shows the type of OPEN resulting from the parameters specified
  - Left side shows the type of data set and type of open
  - Column headings indicate the RLS option specified

Data Set Type & Type of OPEN	NRI	CR	CRE
Recoverable Open for Input	VSAM RLS	VSAM RLS	DFSMStvs
Recoverable Open for Output	DFSMStvs	DFSMStvs	DFSMStvs
Non-recoverable Open for Input	VSAM RLS	VSAM RLS	DFSMStvs
Non-recoverable Open for Output	VSAM RLS	VSAM RLS	DFSMStvs

Figure 7-38 Accessing a data set with DFSMStvs

Data set access with DFSMStvs

In "VSAM RLS locking" on page 366 we talk about the MACRF options NRI, CR, and CRE. CRE gives DFSMStvs access to VSAM data sets open for input or output. CR or NRI gives DFSMStvs access to VSAM recoverable data sets only for output.

You can modify an application to use DFSMStvs by specifying RLS in the JCL or the ACB and having the application access a recoverable data set using either open for input with CRE or open for output from a batch job.

The table in Figure 7-38 shows in which cases DFSMStvs is invoked.

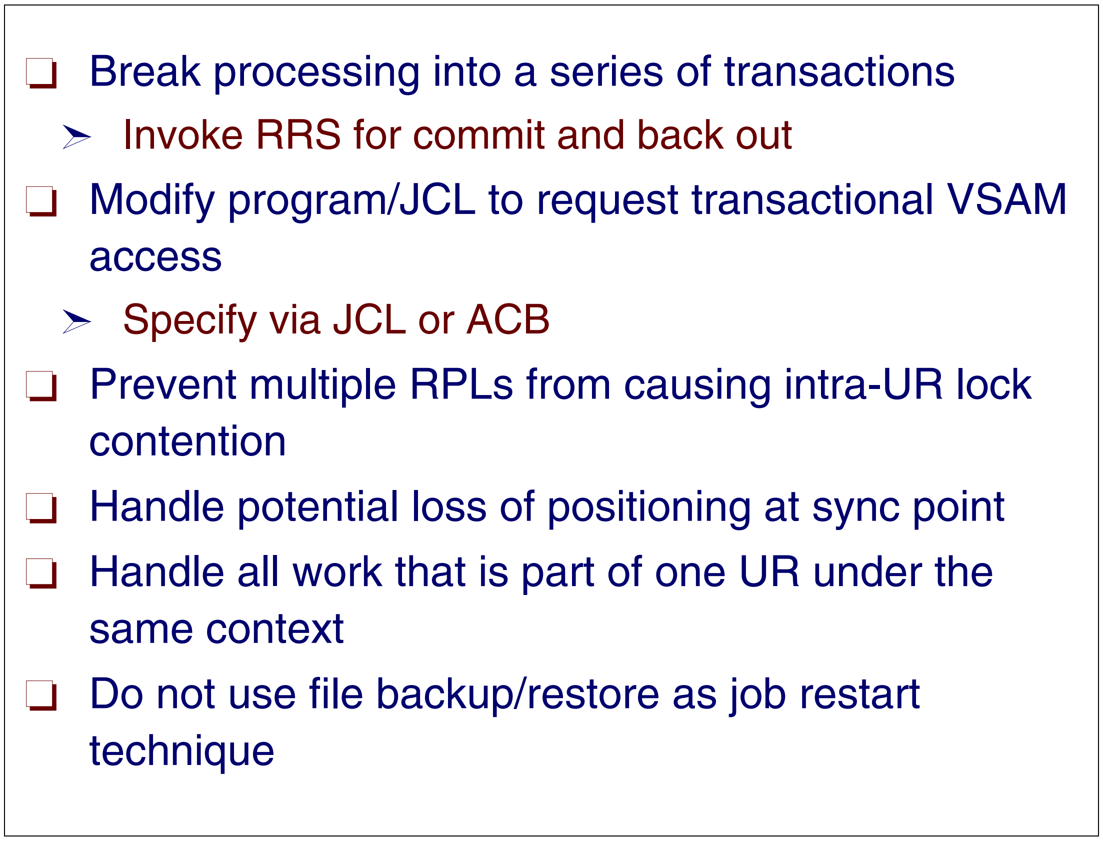
DFSMStvs is always invoked if:

- RLS and CRE is specified in the JCL or MACRF parameter of the ACB macro

  CRE is also known as *repeatable read*.
- RLS is specified in the JCL or MACRF parameter of the ACB macro *and* the data set is recoverable *and* it is opened for output (update processing)

  This allows DFSMStvs to provide the necessary transactional recovery for the data set.

7.30 Application considerations



- Break processing into a series of transactions
  - Invoke RRS for commit and back out
- Modify program/JCL to request transactional VSAM access
  - Specify via JCL or ACB
- Prevent multiple RPLs from causing intra-UR lock contention
- Handle potential loss of positioning at sync point
- Handle all work that is part of one UR under the same context
- Do not use file backup/restore as job restart technique

Figure 7-39 Application considerations

Application considerations

In order for an application to participate in transactional recovery, it must first understand the concept of a transaction. It is *not* a good idea simply to modify an existing batch job to use DFSMStvs with no further changes. This would cause the entire job to be seen as a single transaction. As a result, locks would be held and log records would need to exist for the entire life of the job. This could cause a tremendous amount of contention for the locked resources. It could also cause performance degradation as the undo log becomes exceedingly large.

Break processing into a series of transactions

To exploit the DFSMStvs capabilities, the application processing should be broken down into a series of transactions. The application should issue frequent sync points by invoking RRS commit and backout processing (MVS callable services SRRCMIT and SRRBACK). For information about RRS see "DFSMStvs use of z/OS RRMS" on page 399.

RLS and DFSMStvs provide isolation until commit/backout. The application programmer should consider the following rules:

- Share locks on records accessed with *repeatable read*.
- Hold write locks on changed records until the end of a transaction.
- Use commit to apply all changes and release all locks.

- Information extracted from shared files must not be used across commit/backout for the following reasons:
  - Need to re-access the records
  - Cannot get a record before a sync point and update it after
  - Do not position to a record before a sync point and access it after

Modify the program or JCL to request DFSMStvs access

You need to change the ACB MACRF or JCL specifications for a data set to request DFSMStvs access. See the previous section for more information.

Prevent contention within a unit of recovery

If the application uses multiple RPLs, care must be taken in how they are used. Using different RPLs to access the same record can cause lock contention within a UR.

Handle potential loss of positioning at sync point

The batch application must have a built-in method of tracking its processing position within a series of transactions. One potential method of doing this is to use a VSAM recoverable file to track the job's commit position.

Handle all work that is part of one UR under the same context

For information about units of recovery, see 7.27, “Unit of work and unit of recovery” on page 402. You should reconsider your application to handle work that is part of one unit of recovery under the same context.

Do not use file backup/restore as a job restart technique

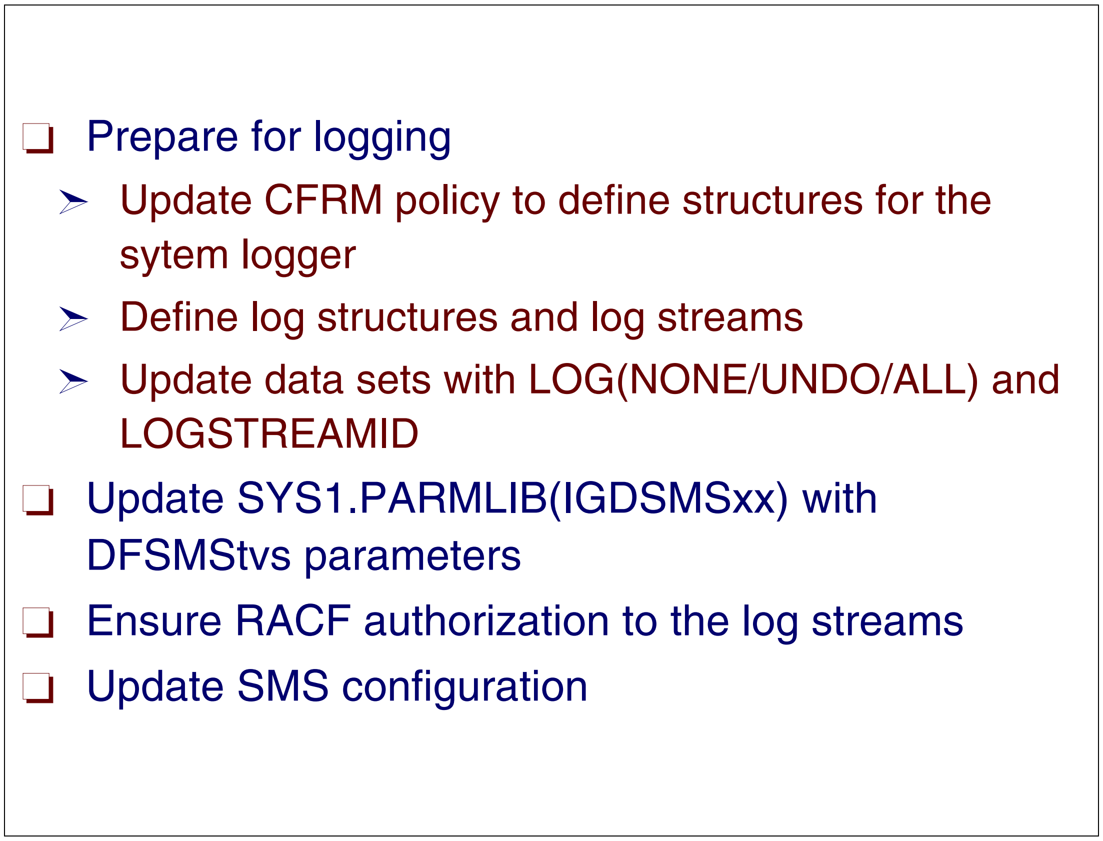
Today's batch applications that update VSAM files in a non-shared environment may create backup copies of the files to establish a restart/recovery point for the data. If the batch application fails, the files may be restored from the backup copies, and the batch jobs can be re-executed. This restart/recovery procedure *cannot* be used in a data sharing DFSMStvs environment because restoring to the point-in-time backup would erase any changes made by other applications sharing the data set.

Instead, the batch application must have a built-in method of tracking its processing position within a series of transactions. One potential method of doing this is to use a VSAM recoverable file to track the job’s commit position. When the application fails, any uncommitted changes are backed out.

The already committed changes cannot be backed out, because they are already visible to other jobs or transactions. In fact, the records that were changed by previously committed UR may have since been changed again by other jobs or transactions. Therefore, when the job is rerun, it is important that it determines its restart point and not attempt to redo any changes it had committed before the failure.

For this reason, it is important that jobs and applications using DFSMStvs be written to execute as a series of transactions and use a commit point tracking mechanism for restart.

7.31 DFSMStvs logging implementation



- ❑ Prepare for logging
  - ➢ Update CFRM policy to define structures for the sytem logger
  - ➢ Define log structures and log streams
  - ➢ Update data sets with LOG(NONE/UNDO/ALL) and LOGSTREAMID
- ❑ Update SYS1.PARMLIB(IGDSMSxx) with DFSMStvs parameters
- ❑ Ensure RACF authorization to the log streams
- ❑ Update SMS configuration

Figure 7-40 DFSMStvs configuration changes

DFSMStvs configuration changes

In order to run DFSMStvs you need to have set up VSAM RLS already. DFSMStvs uses the locking and buffering techniques of VSAM RLS to access the VSAM data sets. Additionally, you need to make the following changes:

- Prepare for logging.
  - Update CFRM policy to define structures for the system logger.
  - Define log streams.
  - Update data sets with LOG(NONE/UNDO/ALL) and LOGSTREAMID.

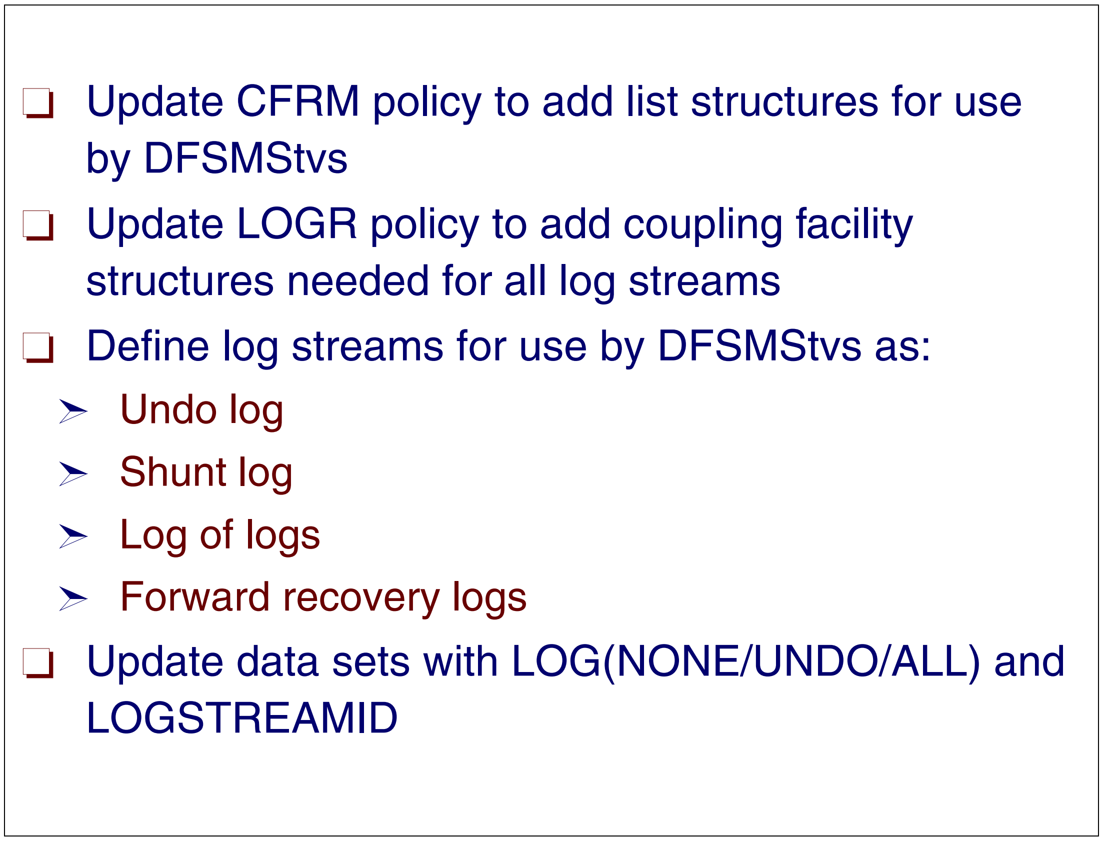
  This is described in more detail in the next section.
- Update SYS1.PARMLIB(IGDSMSxx) with DFSMStvs parameters.

  For the necessary PARMLIB changes see “Update PARMLIB with DFSMStvs parameters” on page 412
- Ensure that DFSMStvs has RACF authorization to the log streams.

  To provide logging, DFSMStvs needs RACF UPDATE authority to the log streams. See also *z/OS DFSMStvs Planning and Operation Guide,* SC26-7348.
- Update SMS configuration.

  Recoverable data sets must be SMS managed. Therefore, you need to change your SMS configuration to assign a storage class to those data sets. In addition, you can optionally change your data class with BWO, LOG, and LOGSTREAMID parameters.

7.32 Prepare for logging



- ❑ Update CFRM policy to add list structures for use by DFSMStvs
- ❑ Update LOGR policy to add coupling facility structures needed for all log streams
- ❑ Define log streams for use by DFSMStvs as:
  - ➢ Undo log
  - ➢ Shunt log
  - ➢ Log of logs
  - ➢ Forward recovery logs
- ❑ Update data sets with LOG(NONE/UNDO/ALL) and LOGSTREAMID

Figure 7-41 Prepare for logging

Prepare for logging

You have to define *log structures* in two places: the coupling facility resource management (CFRM) policy and the system logger LOGR policy. A policy is a couple data set. Furthermore, you need to define *log streams* for the different kinds of logs used by DFSMStvs. If a data set is forward recoverable, you also need to assign a log stream for forward recovery to this data set.

Update the CFRM policy

The CFRM policy is used to divide coupling facility space into structures. The CFRM policy enables you to define how MVS should manage coupling facility resources. In “Coupling facility structures for RLS sharing” on page 373 we describe how to define CF structures for use be VSAM RLS. You need to run a similar job to either define a new CFRM policy or to update an existing CFRM policy with the structures that are used by the logger.

A sample JCL you can use to define a new CFRM policy is shown in Figure 7-42.

```
//LABEL    JOB ...
//STEP10   EXEC PGM=IXCMIAPU
//SYSPRINT DD   SYSOUT=A
//SYSABEND DD   SYSOUT=A
//SYSIN    DD   *
      DATA TYPE(CFRM) REPORT(YES)
      DEFINE POLICY NAME(CFRM02) REPLACE(YES)
        STRUCTURE NAME(LOG_IGWLOG_001)
                  SIZE(10240)
                  INITSIZE(5120)
                  PREFLIST(CF01,CF02)
        STRUCTURE NAME(LOG_IGWSHUNT_001)
                  SIZE(10240)
                  INITSIZE(5120)
                  PREFLIST(CF01,CF02)
        STRUCTURE NAME(LOG_IGWLGLGS_001)
                  SIZE(10240)
                  INITSIZE(5120)
                  PREFLIST(CF01,CF02)
        STRUCTURE NAME(LOG_FORWARD_001)
                  SIZE(10240)
                  INITSIZE(5120)
                  PREFLIST(CF01,CF02)
```

Figure 7-42 Example of defining structures in the CFRM policy

Update the LOGR policy

You must also define the coupling facility structures in the LOGR policy. The system logger component manages log streams based on the policy information that installations place in the LOGR policy.

Multiple log streams can write data to a single coupling facility structure. This does not mean that the log data is merged; the log data stays segregated according to log stream.

Figure 7-43 shows how to define the structures in the LOGR policy.

```
//LABEL    JOB ...
//STEP10   EXEC PGM=IXCMIAPU
//SYSPRINT DD   SYSOUT=A
//SYSABEND DD   SYSOUT=A
//SYSIN    DD   *
      DATA TYPE(LOGR) REPORT(YES)
      DEFINE STRUCTURE NAME(LOG_IGWLOG_001)
                       LOGSNUM(10) MAXBUFSIZE(64000)
                       AVGBUFSIZE(4096)
      DEFINE STRUCTURE NAME(LOG_IGWSHUNT_001)
                       LOGSNUM(10) MAXBUFSIZE(64000)
                       AVGBUFSIZE(4096)
      DEFINE STRUCTURE NAME(LOG_IGWLGLGS_001)
                       LOGSNUM(10) MAXBUFSIZE(64000)
                       AVGBUFSIZE(4096)
      DEFINE STRUCTURE NAME(LOG_FORWARD_001)
                       LOGSNUM(20) MAXBUFSIZE(64000)
                       AVGBUFSIZE(4096)
/*
```

Figure 7-43 Defining structures to the LOGR policy

Define log streams

Each DFSMStvs instance requires its own pair of *system logs*: a primary (undo) log and a secondary (shunt) log. You must define these two logs before you can use DFSMStvs. Additionally, you can define forward recovery logs and a log of logs.

For the different types of log streams that are used by DFSMStvs refer to “DFSMStvs logging” on page 403. A log stream is a VSAM linear data set which simply contains a collection of data. To define log streams, you can use the example JCL in Figure 7-44.

```
//LABEL    JOB ...
//STEP10   EXEC PGM=IXCMIAPU
//SYSPRINT DD   SYSOUT=A
//SYSABEND DD   SYSOUT=A
//SYSIN    DD   *
     DATA TYPE(LOGR) REPORT(YES)
     DEFINE LOGSTREAM NAME(IGWTV001.IGWLOG.SYSLOG)
                      STRUCTURENAME(LOG_IGWLOG_001)
                      LS_SIZE(1180)
                      LS_DATACLAS(dataclas)  LS_STORCLAS(storclas)
                      STG_DUPLEX(YES)        DUPLEXMODE(COND)
                      HIGHOFFLOAD(80)        LOWOFFLOAD(60)
                      DIAG(YES)

     DEFINE LOGSTREAM NAME(IGWTV001.IGWSHUNT.SHUNTLOG)
                      STRUCTURE(LOG_IGWSHUNT_001)
                      LS_SIZE(100)
                      LS_DATACLAS(dataclas) LS_STORCLAS(storclas)
                      HIGHOFFLOAD(80)       LOWOFFLOAD(0)
                      DIAG(YES)
/
```

Figure 7-44 Example of defining log streams

Defining log streams

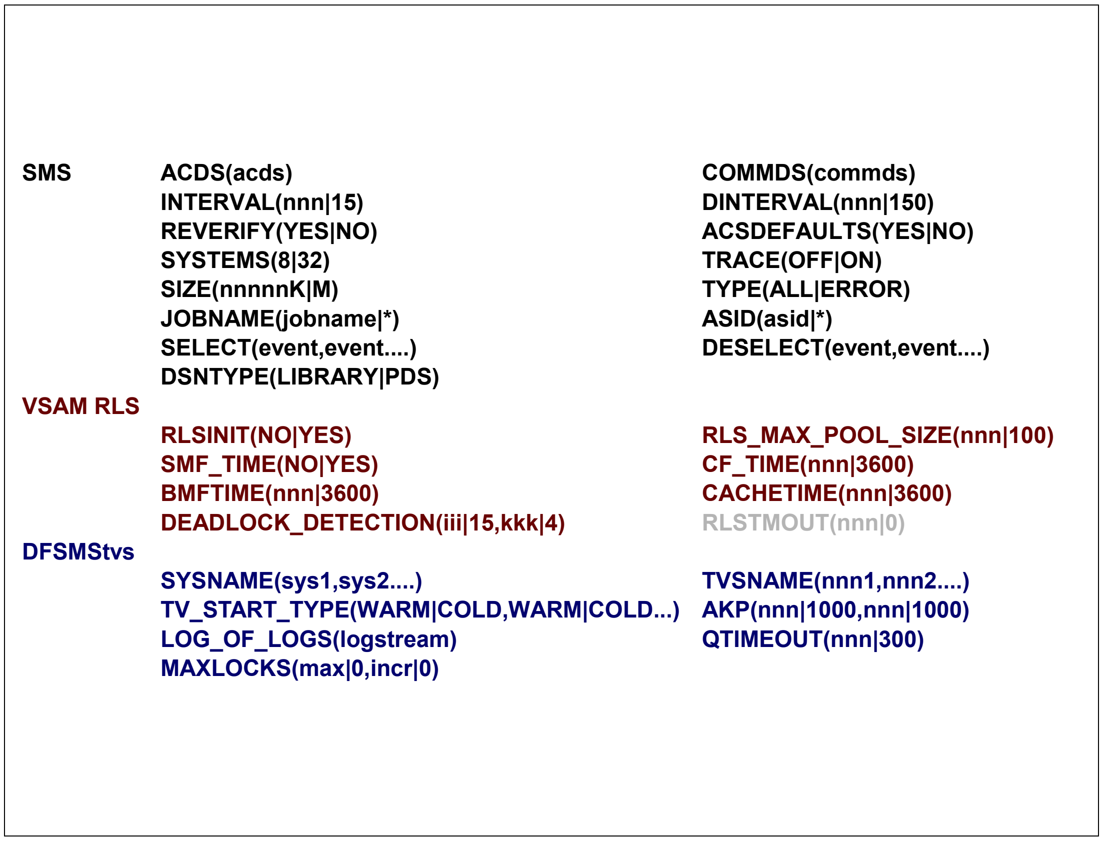
Each log stream is assigned to a structure previously defined to the LOGR policy. You can assign multiple log streams to the same structure, for example, for the forward recovery logs. If you were using CICS forward recovery in the past, you don’t need to define new log streams for forward recovery. CICS and DFSMStvs can share the same forward recovery logs.

Attention: Log streams are single-extent VSAM linear data sets and need SHAREOPTIONS(3,3). The default is SHAREOPTIONS(1,3), so you must alter the share options explicitly by running IDCAMS ALTER.

Update data set attributes

You need to update your VSAM data sets with the LOG and LOGSTREAMID parameters in order to use DFSMStvs. If you were using VSAM RLS already and don’t want to change the kind of recovery, no changes are necessary. See also “Update data sets with log parameters” on page 384.

7.33 Update PARMLIB with DFSMStvs parameters



SMS	ACDS(acds)	COMMDS(commds)
	INTERVAL(nnn|15)	DINTERVAL(nnn|150)
	REVERIFY(YES|NO)	ACSDEFAULTS(YES|NO)
	SYSTEMS(8|32)	TRACE(OFF|ON)
	SIZE(nnnnnK|M)	TYPE(ALL|ERROR)
	JOBNAME(jobname|*)	ASID(asid|*)
	SELECT(event,event....)	DESELECT(event,event....)
	DSNTYPE(LIBRARY|PDS)	
VSAM RLS		
	RLSINIT(NO|YES)	RLS_MAX_POOL_SIZE(nnn|100)
	SMF_TIME(NO|YES)	CF_TIME(nnn|3600)
	BMFTIME(nnn|3600)	CACHETIME(nnn|3600)
	DEADLOCK_DETECTION(iii|15,kkk|4)	RLSTMOUT(nnn|0)
DFSMStvs		
	SYSNAME(sys1,sys2....)	TVSNAME(nnn1,nnn2....)
	TV_START_TYPE(WARM|COLD,WARM|COLD...)	AKP(nnn|1000,nnn|1000)
	LOG_OF_LOGS(logstream)	QTIMEOUT(nnn|300)
	MAXLOCKS(max|0,incr|0)	

Figure 7-45 PARMLIB parameters to support DFSMStvs

New PARMLIB parameters to support DFSMStvs

There are a few new parameters you can specify in the PARMLIB member IGDSMSxx to support DFSMStvs. Please note, that DFSMStvs requires VSAM RLS. For a description of the VSAM RLS parameters refer to “Update PARMLIB with VSAM RLS parameters” on page 376.

These are the new parameters for DFSMStvs:

- SYSNAME(sysname[,sysname]...)
  - Identifies the systems on which you want to run DFSMStvs.
  - Specifies the names of the systems to which the DFSMStvs instance names of the TVSNAME parameter apply.
- TVSNAME(nnn[,nnn]...)
  - Specifies the identifiers of the DFSMStvs instances on the systems you specified in the SYSNAME parameter.
  - If only one TVSNAME is specified, this parameter applies only to the system on which the PARMLIB member is read; in this case, no SYSNAME is required.
  - Number of sysnames and number of tvsnames must be the same.

- TV_START_TYPE({WARM|COLD}[,{WARM|COLD}]...)
  - Specifies the start type for the single DFSMStvs instances as they are listed in the TVSNAME parameter.
- AKP(nnn[,nnn]...)
  - Specifies the activity keypoint trigger values for the DFSMStvs instances as they appear in the TVSNAME parameter.
  - AKP is the number of logging operations between keypoints.
  - Activity keypointing enables DFSMStvs to delete records from the undo or shunt log that are no longer involved in active units of recovery.
- LOG_OF_LOGS(logstream)
  - Specifies the name of the log stream which is used as log of logs.
- QTIMEOUT({nnn|300})
  - Specifies the amount of time the DFSMStvs quiesce exits allow to elapse before concluding that a quiesce event cannot be completed successfully.
- MAXLOCKS({max|0},{incr|0})
  - *max* is the maximum number of unique lock requests that a single unit of recovery can make; when this value is reached, the system will issue a warning message.
  - *incr* is an increment value.
  - After *max* is reached, each time the number of locks increases by the *incr* value, another warning message is issued.

For information about these PARMLIB parameters, refer to *z/OS MVS Initialization and Tuning Reference,* SA22-7592.

7.34 The DFSMStvs instance

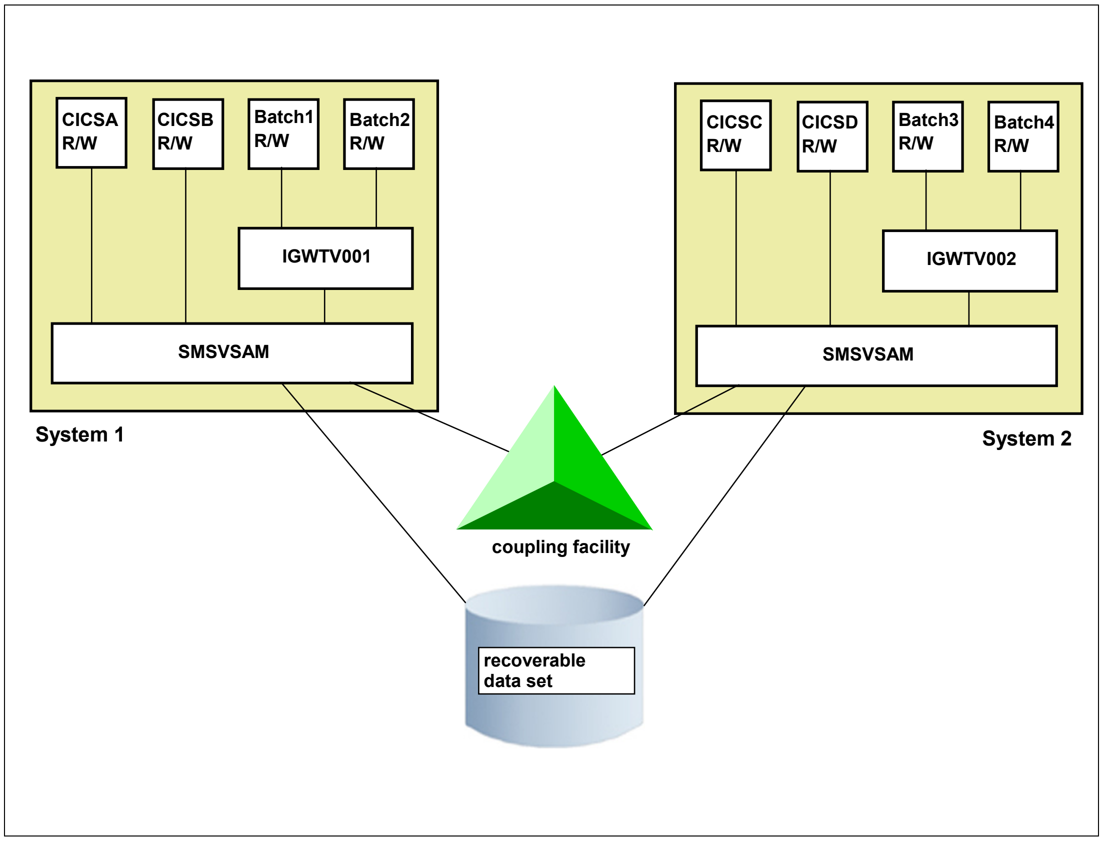


Figure 7-46 Example of a sysplex with two active DFSMStvs instances

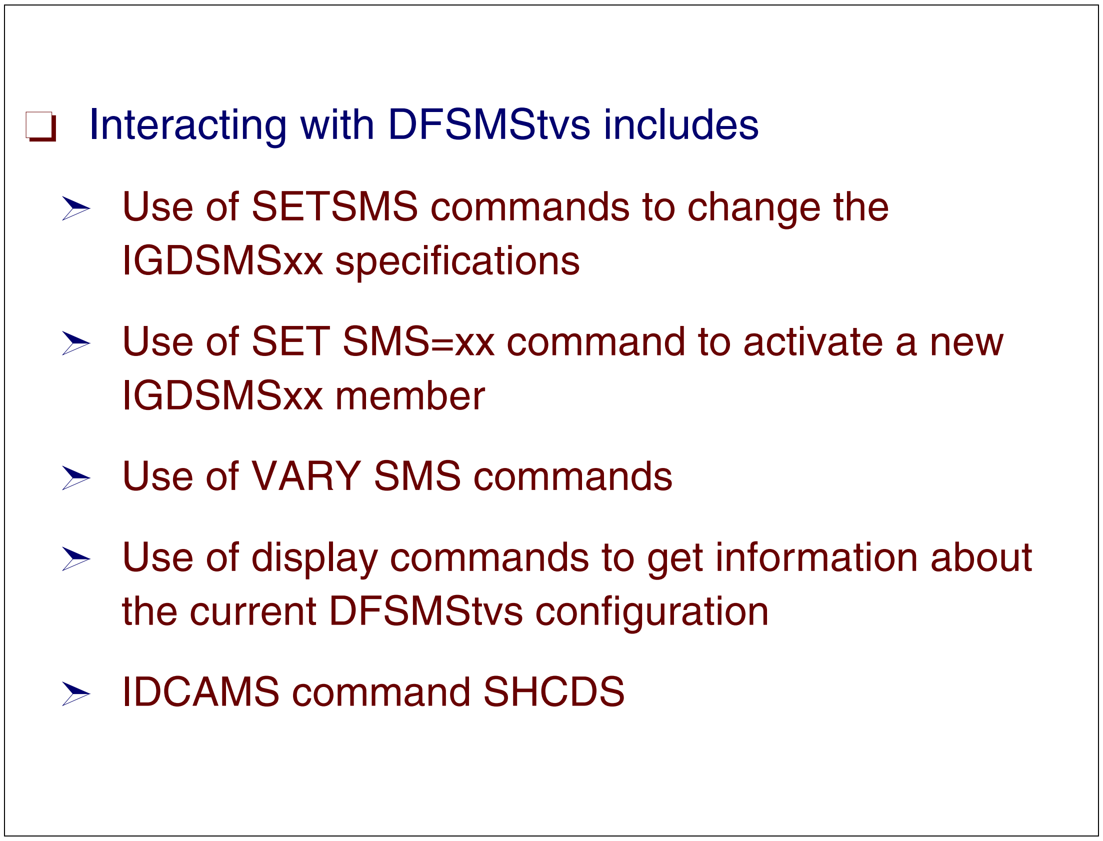
The DFSMStvs instance

DFSMStvs runs in the SMSVSAM address space as an instance. The name of the SMSVSAM address space is always IGWTV*nnn*, where *nnn* is a unique number per system as defined for TVSNAME in the PARMLIB member IGDSMSxx. This number can range from 0 to 255. You can define up to 32 DFSMStvs instances, one per system. An example is IGWTV001.

As soon as an application that does not act as a recoverable resource manager has RLS access to a recoverable data set, DFSMStvs is invoked (see also “Accessing a data set with DFSMStvs” on page 405). DFSMStvs calls VSAM RLS (SMSVSAM) for record locking and buffering. With DFSMStvs built on top of VSAM RLS, full sharing of recoverable files becomes possible. Batch jobs can now update the recoverable files without first quiescing CICS' access to them.

As a recoverable resource manager, CICS interacts directly with VSAM RLS.

7.35 Interacting with DFSMStvs



- Interacting with DFSMStvs includes
  - Use of SETSMS commands to change the IGDSMSxx specifications
  - Use of SET SMS=xx command to activate a new IGDSMSxx member
  - Use of VARY SMS commands
  - Use of display commands to get information about the current DFSMStvs configuration
  - IDCAMS command SHCDS

Figure 7-47 Operator commands to interact with DFSMStvs

Interacting with DFSMStvs

In this section we give a short overview of some commands that were introduced particularly for displaying and changing DFSMStvs related information. You can use those commands in addition to the commands we already described in 7.20, "Interacting with VSAM RLS" on page 388.

SETSMS command

Use the **SETSMS** command to overwrite the PARMLIB specifications for IGDSMSxx. The syntax is:

```
SETSMS AKP(nnn|1000)
       QTIMEOUT(nnn|300)
       MAXLOCKS(max|0,incr|0)
```

These are the only DFSMStvs PARMLIB specifications you can overwrite using the SETSMS command. For information about these parameter see "Update PARMLIB with DFSMStvs parameters" on page 412.

SET SMS=xx command

The command **SET SMS=*xx*** causes the IGDSMSxx member of SYS1.PARMLIB to be read. It allows you to activate another IGDSMSxx member, where ***xx*** specifies the last two digits of the member. You can change DFSMStvs parameters in IGDSMSxx that you can't change with the **SETSMS** command, and activate the changes afterwards by running **SET SMS=xx**. Changes to parameters other than AKP, QTIMEOUT and MAXLOCKS take affect the next time DFSMStvs restarts.

VARY SMS command

The **VARY SMS** command has the following functions for DFSMStvs:

- Enable, quiesce, or disable one or all DFSMStvs instances:

  ```
  VARY SMS,TRANVSAM(tvsname|ALL),{QUIESCE|Q}
                                 {DISABLE|D}
                                 {ENABLE|E}
  ```

  This command is routed to all systems in the sysplex. If you specify ALL it effects all DFSMStvs instances; otherwise, it effects only the instance specified by *tvsname*. If QUIESCE is specified, DFSMStvs completes the current work first, but does not accept any new work. If DISABLE is specified, DFSMStvs stops processing immediately.

- Enable, quiesce, or disable DFSMStvs access to a specified logstream:

  ```
  VARY SMS,LOG(logstream),{QUIESCE|Q}
                          {DISABLE|D}
                          {ENABLE|E}
  ```

  Quiescing or disabling the DFSMStvs undo or shunt logstream is equivalent to quiescing or disabling DFSMStvs processing respectively. Quiescing or disabling the log of logs has no influence on DFSMStvs processing. Quiescing or disabling a forward recovery log causes all attempts to process data sets which use this log stream to fail.

- Start or stop peer recovery processing for a failed instance of DFSMStvs:

  ```
  VARY SMS,TRANVSAM(tvsname),PEERRECOVERY,{ACTIVE}
                                          {ACTIVEFORCE}
                                          {INACTIVE}
  ```

  This command applies only to the system on which it is issued. That system will then be responsible for performing all peer recovery processing for a failed DFSMStvs instance. For a discussion of the term *peer recovery*, see 7.24, "Overview of DFSMStvs" on page 397.

Display command

There are a few display commands you can use to get information about DFSMStvs.

- Display common DFSMStvs information:

  ```
  DISPLAY SMS,TRANVSAM{,ALL}
  ```

  This command lists information about the DFSMStvs instance on the system were it was issued. To get information from all systems use ALL. This information includes name and state of the DFSMStvs instance, values for AKP, start type, and qtimeout, and also the names, types, and states of the used log streams.

- Display information about a particular job that uses DFSMStvs:

  ```
  DISPLAY SMS,JOB(jobname)
  ```

  The information about the particular job includes the current job step, the current ID, and status of the unit of recovery used by this job.

- Display information about a particular unit of recovery currently active within the sysplex:

  ```
  DISPLAY SMS,URID(urid|ALL)
  ```

  This command provides information about a particular UR in the sysplex or about all URs of the system on which this command was issued. If ALL is specified, you don't get information about shunted URs and URs that are restarting. The provided information includes age and status of the UR, the jobname with which this UR is associated, and the current step within the job.

- Display entries currently contained in the shunt log:

  ```
  DISPLAY SMS,SHUNTED,{SPHERE(sphere)|
                      URID(urid|ALL)}
  ```

  Entries are moved to the shunt log when DFSMStvs is unable to finish processing a sync point for them. There are several reasons this might occur; an I/O error is one of the possible causes. Depending on what was specified, you get information for a particular VSAM sphere, a particular UR, or for all shunted URs in the sysplex.

- Display information about logstreams that DFSMStvs is currently using:

  ```
  DISPLAY SMS,LOG(logstream|ALL)
  ```

  If ALL is specified, information about all log streams in use is provided from the system on which the command is issued. The output includes the status and type of the log stream, the job name and URID of the oldest unit of recovery using the log, and also a list of all DFSMStvs instances that are using the log.

- Display information about a particular data set:

  ```
  DISPLAY SMS,DSNAME(dsn)
  ```

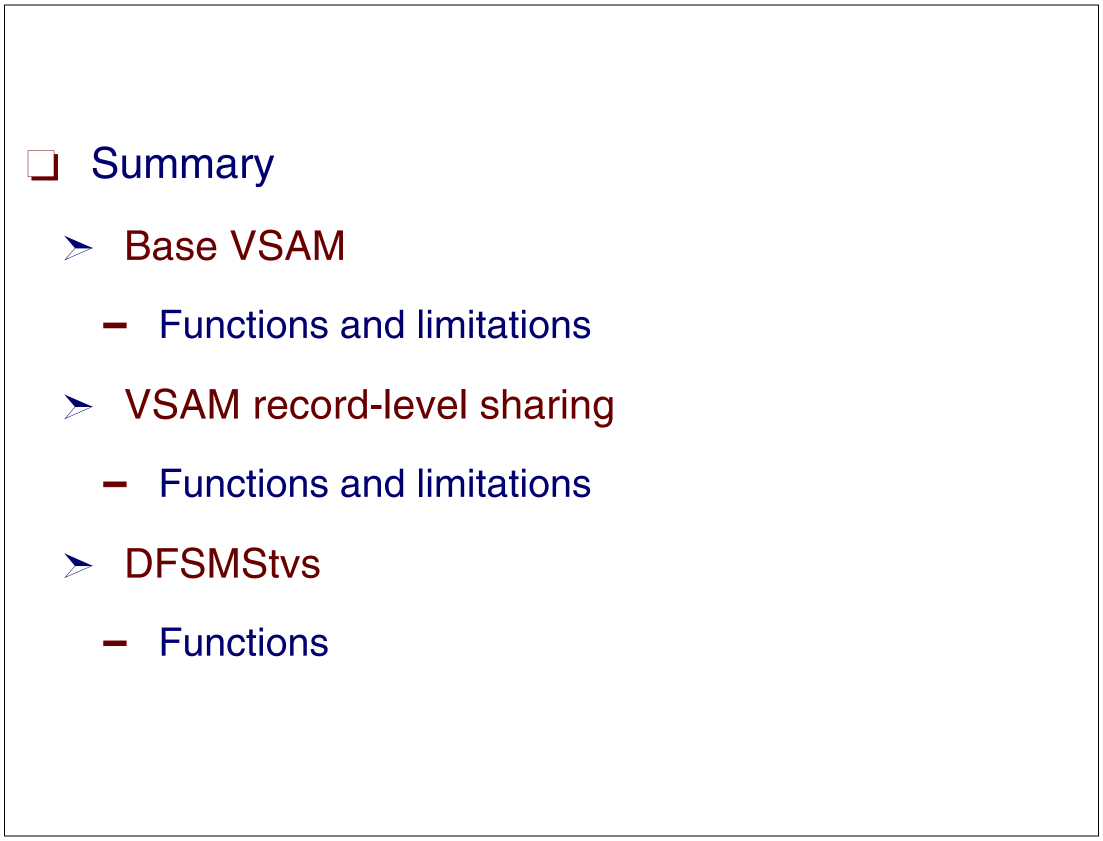
  Use this command to display jobs currently accessing the data set using DFSMStvs access on the systems within the sysplex.

IDCAMS command SHCDS

The IDCAMS command SHCDS was enhanced to display and modify recovery information related to DFSMStvs. For more information about this command refer to *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394.

Attention: This chapter provides only an overview about new operator commands you should know to work with DFSMStvs. Before you use any of these commands other than the **`DISPLAY`** command, read the official z/OS manuals carefully.

7.36 Summary



❑ Summary

➢ Base VSAM

– Functions and limitations

➢ VSAM record-level sharing

– Functions and limitations

➢ DFSMStvs

– Functions

Figure 7-48

Summary

In this chapter we showed the limitations of base VSAM that made it necessary to develop VSAM RLS. Further, we exposed the limitations of VSAM RLS that were the reason to enhance VSAM RLS by the functions provided by DFSMStvs.

- Base VSAM
  - VSAM does not provide read or read/write integrity for share options other than 1.
  - User needs to use enqueue/dequeue macros for serialization.
  - The granularity of sharing on a VSAM cluster is at the control interval level.
  - Buffers reside in the address space.
  - Base VSAM does not support CICS as a recoverable resource manager; a CICS file owning region is necessary to ensure recovery.
- VSAM RLS
  - Enhancement of base VSAM.
  - User does not need to serialize; this is done by RLS locking.
  - Granularity of sharing is record level, not CI level.
  - Buffers reside in the data space and coupling facility.
  - Supports CICS as a recoverable resource manager (CICS logging for recoverable data sets); no CICS file owning region is necessary.

- Does not act as a recoverable resource manager (provides no logging for recoverable data sets).
- Sharing of recoverable data sets that are in use by CICS with non-CICS applications like batch jobs is not possible.

- DFSMStvs
  - Enhancement of VSAM RLS.
  - Uses VSAM RLS to access VSAM data sets.
  - Acts as a recoverable resource manager, thus provides logging.
  - Enables non-CICS applications like batch jobs to share recoverable data sets with CICS.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 422. Note that some of the documents referenced here may be available in softcopy only.

- *VSAM Demystified*, SG24-6105
- *DFSMStvs Overview and Planning Guide*, SG24-6971
- *DFSMStvs Presentation Guide,* SG24-6973
- *z/OS DFSMS V1R3 and V1R5 Technical Guide,* SG24-6979

Other publications

These publications are also relevant as further information sources:

- *z/OS DFSMStvs Administration Guide,* GC26-7483
- *Device Support Facilities User’s Guide and Reference Release 17*, GC35-0033
- *z/OS MVS Programming: Assembler Services Guide,* SA22-7605
- *z/OS MVS System Commands,* SA22-7627
- *z/OS MVS System Messages,Volume 1 (ABA-AOM),* SA22-7631
- *DFSMS Optimizer User’s Guide and Reference,* SC26-7047
- *z/OS DFSMStvs Planning and Operating Guide,* SC26-7348
- *z/OS DFSMS Access Method Services for Catalogs,* SC26-7394
- *z/OS DFSMSdfp Storage Administration Reference,* SC26-7402
- *z/OS DFSMSrmm Guide and Reference,* SC26-7404
- *z/OS DFSMSrmm Implementation and Customization Guide,* SC26-7405
- *z/OS DFSMS Implementing System-Managed Storage,* SC26-7407
- *z/OS DFSMS: Using Data Sets,* SC26-7410
- *z/OS DFSMS: Using the Interactive Storage Management Facility,* SC26-7411
- *z/OS DFSMS: Using Magnetic Tapes*, SC26-7412
- *z/OS DFSMSdfp Utilities,* SC26-7414
- *z/OS Network File System Guide and Reference*, SC26-7417
- *DFSORT Getting Started with DFSORT R14,* SC26-4109
- *DFSORT Installation and Customization Release 14,* SC33-4034
- *z/OS DFSMShsm Storage Administration Guide*, SC35-0421
- *z/OS DFSMShsm Storage Administration Reference,* SC35-0422

- *z/OS DFSMSdss Storage Administration Guide*, SC35-0423
- *z/OS DFSMSdss Storage Administration Reference,* SC35-0424
- *z/OS DFSMS Object Access Method Application Programmer's Reference,* SC35-0425
- *z/OS DFSMS Object Access Method Planning, Installation, and Storage Administration Guide for Object Support,* SC35-0426
- *Tivoli Decision Support for OS/390 System Performance Feature Reference Volume I,* SH19-6819

Online resources

These Web sites and URLs are also relevant as further information sources:

- For articles, online books, news, tips, techniques, examples, and more, visit the z/OS DFSORT home page:

  `http://www-1.ibm.com/servers/storage/support/software/sort/mvs`

For more information about DFSMSrmm, visit:

`http://www-1.ibm.com/servers/storage/software/sms/rmm/`

How to get IBM Redbooks publications

You can search for, view, or download books, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy books or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM

Redbooks

ABCs of z/OS System Programming Volume 3

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

ABCs of z/OS System Programming Volume 3

DFSMS, Data set basics, SMS

Storage management software and hardware

Catalogs, VSAM, DFSMStvs

The ABCs of z/OS System Programming is an eleven volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects. If you would like to become more familiar with z/OS in your current environment, or if you are evaluating platforms to consolidate your e-business applications, the ABCs collection will serve as a powerful technical tool.

The contents of the volumes are:

- Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, and z/OS delivery and installation
- Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, LNKLST, authorized libraries, Language Environment, and SMP/E
- Volume 3: Introduction to DFSMS, data set basics, storage management hardware and software, VSAM, System-managed storage, catalogs, and DFSMStvs
- Volume 4: Communication Server, TCP/IP and VTAM
- Volume 5: Base and Parallel Sysplex, System Logger, Resource Recovery Services (RRS), global resource serialization (GRS), z/OS system operations, automatic restart management (ARM), Geographically dispersed Parallel Sysplex (GPDS)
- Volume 6: Introduction to security, RACF, Digital certificates and PKI, Kerberos, cryptography and z990 integrated cryptography, zSeries firewall technologies, LDAP, Enterprise identity mapping (EIM), and firewall technologies
- Volume 7: Printing in a z/OS environment, Infoprint Server and Infoprint Central
- Volume 8: An introduction to z/OS problem diagnosis
- Volume 9: z/OS UNIX System Services
- Volume 10: Introduction to z/Architecture, zSeries processor design, zSeries connectivity, LPAR concepts, HCD, and HMC
- Volume 11: Capacity planning, performance management, WLM, RMF, and SMF

SG24-6983-02 ISBN 0738489050